[SQL UNION Operation - Tutorial Republic](https://www.tutorialrepublic.com/sql-tutorial/sql-union-operation.php)

**SQL Getting Started**

In this tutorial you will learn how to set up a SQL practice environment.

**Getting Started with SQL**

As you already know SQL is used to communicate with the database, so before you start experimenting with SQL, you need access to a database system first.

You can test or execute most of the SQL statements provided as examples throughout the tutorials, using our [online SQL editor](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=select-all). This SQL editor uses Web SQL Database to store and access data on the client side. However, to execute some SQL statement you'll need access to a full-fledged database management system like MySQL, SQL Server etc.

**Note:** Web SQL Database allows you to create SQL databases and make SQL calls on the client side. It is based on the popular and open source [SQLite](https://sqlite.org/) engine. Web SQL is supported by Chrome, Opera and Safari browsers.

**What is Relational Database**

A relational database is a database divided into logical units called tables, where tables are related to one another within the database. Relational database allows data to be broken down into logical, smaller, and manageable units for easier maintenance and better performance.

Tables are related to one another through common keys or fields in a relational database system, that's why even though the desired data may exist in more than one table, you can easily join multiple tables together to get combined data set using a single query.

**Note:** SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. Although, most SQL code is not completely portable among different database systems without adjustments.

**Setting Up Work Environment for Practicing SQL**

If you have no DBMS (Database Management System) already installed on your system, you have several options to choose from. You can install a free, open-source DBMS.

MySQL is the most popular and widely supported open-source database management system. It is very easy to download and use and available for both Windows and Linux (or UNIX) operating system. You can download it freely from here <https://dev.mysql.com/downloads/mysql/>

You can also install SQL Server Express. It is a free version of Microsoft SQL Server which allows up to 10GB of database storage. You can download the express edition from here <https://www.microsoft.com/en-in/download/details.aspx?id=30438>.

Alternatively, if you're planning to develop an application with PHP and MySQL you can install WampServer or XAMPP. WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database. It will also provide the MySQL administrative tool PhpMyAdmin to easily manage your databases using a web browser. You can download it from here <http://www.wampserver.com/en/>

**SQL Syntax**

The syntax of SQL is governed by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO).

**SQL Statements**

SQL statements are very simple and straightforward like plain English but with specific syntax.

An SQL statement is composed of a sequence of keywords, identifiers, etc. terminated by a semicolon (;). Here is an example of a valid SQL statement.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=single-line-syntax)

SELECT emp\_name, hire\_date, salary FROM employees WHERE salary > 5000;

For better readability you can also write the same statement, as follow:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=line-breaks-inside-sql-query)

SELECT emp\_name, hire\_date, salary

FROM employees

WHERE salary > 5000;

Use semicolon at the end of an SQL statement — it terminates the statement or submits the statement to the database server. Some database management system has, however, no such requirement, but it is considered as a best practice to use it.

We'll discuss each part of these statements in detail in upcoming chapters.

**Note:** Any number of line breaks may occur within a SQL statement, provided that any line break does not break off keywords, values, expression, etc.

**Case Sensitivity in SQL**

Consider another SQL statement that retrieves the records from *employees* table:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=uppercase-syntax)

SELECT emp\_name, hire\_date, salary FROM employees;

The same statement can also be written, as follow:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=lowercase-syntax)

select emp\_name, hire\_date, salary from employees;

SQL keywords are case-insensitive that means SELECT is same as select. But, the database and table names may case-sensitive depending on the operating system. In general, Unix or Linux platforms are case-sensitive, whereas Windows platforms aren't.

**Tip:** It is recommended to write the SQL keywords in uppercase, to differentiate it from other text inside a SQL statement for a better understanding.

**SQL Comments**

A comment is simply a text that is ignored by the database engine. Comments can be used to provide a quick hint about the SQL statement.

SQL support single-line as well as multi-line comments. To write a single-line comment start the line with two consecutive hyphens (--). For example:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=single-line-comment)

-- Select all the employees

SELECT \* FROM employees;

However to write multi-line comments, start the comment with a slash followed by an asterisk (/\*) and end the comment with an asterisk followed by a slash (\*/), like this:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=multiple-line-comment)

/\* Select all the employees whose

salary is greater than 5000 \*/

SELECT \* FROM employees

WHERE salary > 5000;

[PREVIOUS](https://www.tutorialrepublic.com/sql-tutorial/sql-get-started.php)

**SQL CREATE DATABASE Statement**

In this tutorial you will learn how to create database in a relational database management system like, MySQL, SQL Server, etc. using SQL.

**Creating a Database**

Before doing anything with the data we must need to create a database first. We're assuming that you already have a MySQL, or SQL Server available for your use, as well as you've all the necessary privileges, if not please check out the [getting started guide](https://www.tutorialrepublic.com/sql-tutorial/sql-get-started.php).

The SQL CREATE DATABASE statement is used to create a database.

**Syntax**

The basic syntax for creating a database can be given with:

CREATE DATABASE *database\_name*;

The following SQL statement creates a database named *demo*:

CREATE DATABASE demo;

Creating a database does not select it for use. So, before moving further we must need to select the target database with the USE statement. For example, the USE demo; command sets the *demo* database as target database for all future commands.

**Note:** In Unix, the database and table names are case-sensitive, so you must always refer to your database as demo, not as Demo, DEMO, or something else. But, SQL keywords are case-insensitive, like CREATE DATABASE is same as create database.

**Creating Database in MySQL**

Let's create a database in MySQL using the command-line tool.

**Step 1: Invoke the MySQL command-line tool**

To invoke the MySQL command line, we've to log in to the MySQL server first. To log in as root user, type the following command in terminal and press enter. You will be asked for your password. Enter your password and press enter, if it is correct the mysql> prompt will appear, via which you will be able to issue SQL statements and view the results.

shell> mysql -u root -p

**Step 2: Creating a MySQL Database**

Now, execute the following command to create the database named *demo*.

mysql> CREATE DATABASE demo;

If the database created successfully you'll see the output something like this:

Query OK, 1 row affected (0.03 sec)

If you try to create a database that is already exists you'll get an error message. To avoid this in MySQL you can use an optional clause IF NOT EXISTS as follow:

mysql> CREATE DATABASE IF NOT EXISTS demo;

**Step 3: Selecting the Database**

Type the following command and press enter. You will see the output *"Database changed"*. Now our *demo* database is selected as default database for all future operations.

mysql> USE demo;

**SQL CREATE TABLE Statement**

In this tutorial you will learn how to create a table inside the database using SQL.

**Creating a Table**

In the previous chapter we have learned how to create a database on the database server. Now it's time to create some tables inside our database that will actually hold the data. A database table simply organizes the information into rows and columns.

The SQL CREATE TABLE statement is used to create a table.

**Syntax**

The basic syntax for creating a table can be given with:

CREATE TABLE *table\_name* (

*column1\_name data\_type constraints,*

*column2\_name data\_type constraints,*

....

);

To understand this syntax easily, let's create a table in our *demo* database. Type the following statement on MySQL command-line tool and press enter:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=create-table)

-- Syntax for MySQL Database

CREATE TABLE persons (

id INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

birth\_date DATE,

phone VARCHAR(15) NOT NULL UNIQUE

);

-- Syntax for SQL Server Database

CREATE TABLE persons (

id INT NOT NULL PRIMARY KEY IDENTITY(1,1),

name VARCHAR(50) NOT NULL,

birth\_date DATE,

phone VARCHAR(15) NOT NULL UNIQUE

);

The above statement creates a table named *persons* with four columns *id*, *name*, *birth\_date* and *phone*. Notice that each column name is followed by a data type declaration; this declaration specifies that what type of data the column will store, whether integer, string, date, etc.

Some [data types](https://www.tutorialrepublic.com/sql-reference/mysql-data-types.php) can be declared with a length parameter that indicates how many characters can be stored in the column. For example, VARCHAR(50) can hold up to 50 characters.

**Note:** The data type of the columns may vary depending on the database system. For example, MySQL and SQL Server supports INT data type for integer values, whereas the Oracle database supports NUMBER data type.

The following table summarizes the most commonly used data types supported by MySQL.

| **Data Type** | **Description** |
| --- | --- |
| INT | Stores numeric values in the range of -2147483648 to 2147483647 |
| DECIMAL | Stores decimal values with exact precision. |
| CHAR | Stores fixed-length strings with a maximum size of 255 characters. |
| VARCHAR | Stores variable-length strings with a maximum size of 65,535 characters. |
| TEXT | Stores strings with a maximum size of 65,535 characters. |
| DATE | Stores date values in the YYYY-MM-DD format. |
| DATETIME | Stores combined date/time values in the YYYY-MM-DD HH:MM:SS format. |
| TIMESTAMP | Stores timestamp values. [TIMESTAMP](https://www.tutorialrepublic.com/sql-reference/mysql-data-types.php#datetime-and-timestamp) values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:01' UTC). |

Please check out the reference section [SQL DB data types](https://www.tutorialrepublic.com/sql-reference/mysql-data-types.php) for the detailed information on all the data types available in popular RDBMS like MySQL, SQL Server, etc.

There are a few additional constraints (also called *modifiers*) that are set for the table columns in the preceding statement. Constraints define rules regarding the values allowed in columns.

* The NOT NULL constraint ensures that the field cannot accept a NULL value.
* The PRIMARY KEY constraint marks the corresponding field as the table's primary key.
* The AUTO\_INCREMENT attribute is a MySQL extension to standard SQL, which tells MySQL to automatically assign a value to this field if it is left unspecified, by incrementing the previous value by 1. Only available for numeric fields.
* The UNIQUE constraint ensures that each row for a column must have a unique value.

We will learn more about the [SQL constraints](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php) in next chapter.

**Note:** The Microsoft SQL Server uses the IDENTITY property to perform an auto-increment feature. The default value is IDENTITY(1,1) which means the seed or starting value is 1, and the incremental value is also 1.

**Tip:** You can execute the command DESC *table\_name*; to see the column information or structure of any table in MySQL and Oracle database, whereas EXEC sp\_columns *table\_name*; in SQL Server (replace the *table\_name* with actual table name).

**Create Table If Not Exists**

If you try to create a table that is already exists inside the database you'll get an error message. To avoid this in MySQL you can use an optional clause IF NOT EXISTS as follow:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=create-table-if-not-exists)

CREATE TABLE IF NOT EXISTS persons (

id INT NOT NULL PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

birth\_date DATE,

phone VARCHAR(15) NOT NULL UNIQUE

);

**Tip:** If you want to see the list of tables inside the currently selected database, you can execute SHOW TABLES; statement on the MySQL command line.

**SQL Constraints**

In this tutorial you will learn how to use SQL constraints.

**What is Constraint?**

A constraint is simply a restriction placed on one or more columns of a table to limit the type of values that can be stored in that column. Constraints provide a standard mechanism to maintain the accuracy and integrity of the data inside a database table.

There are several different types of constraints in SQL, including:

* [NOT NULL](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php#not-null)
* [PRIMARY KEY](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php#primary-key)
* [UNIQUE](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php#unique)
* [DEFAULT](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php#default)
* [FOREIGN KEY](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php#foreign-key)
* [CHECK](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php#check)

Now, let's discuss each of these constraints in detail.

**NOT NULL Constraint**

The NOT NULL constraint specifies that the column does not accept NULL values.

This means if NOT NULL constraint is applied on a column then you cannot insert a new row in the table without adding a non-NULL value for that column.

The following SQL statement creates a table named *persons* with four columns, out of which three columns, *id*, *name* and *phone* do not accept NULL values.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=not-null-constraint)

CREATE TABLE persons (

id INT NOT NULL,

name VARCHAR(30) NOT NULL,

birth\_date DATE,

phone VARCHAR(15) NOT NULL

);

**Note:** A null value or NULL is different from zero (0), blank, or a zero-length character string such as ''. NULL means that no entry has been made.

**PRIMARY KEY Constraint**

The PRIMARY KEY constraint identify the column or set of columns that have values that uniquely identify a row in a table. No two rows in a table can have the same primary key value. Also, you cannot enter NULL value in a primary key column.

The following SQL statement creates a table named *persons* and specifies the *id* column as the primary key. That means this field does not allow NULL or duplicate values.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=primary-key-constraint)

CREATE TABLE persons (

id INT NOT NULL PRIMARY KEY,

name VARCHAR(30) NOT NULL,

birth\_date DATE,

phone VARCHAR(15) NOT NULL

);

**Tip:** The primary key typically consists of one column in a table, however more than one column can comprise the primary key, e.g. either the employee's email address or assigned identification number is the logical primary key for an employee table.

**UNIQUE Constraint**

The UNIQUE constraint restricts one or more columns to contain unique values within a table.

Although both a UNIQUE constraint and a PRIMARY KEY constraint enforce uniqueness, use a UNIQUE constraint instead of a PRIMARY KEY constraint when you want to enforce the uniqueness of a column, or combination of columns, that is not the primary key.

The following SQL statement creates a table named *persons* and specifies the *phone* column as unique. That means this field does not allow duplicate values.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=unique-constraint)

CREATE TABLE persons (

id INT NOT NULL PRIMARY KEY,

name VARCHAR(30) NOT NULL,

birth\_date DATE,

phone VARCHAR(15) NOT NULL UNIQUE

);

**Note:** Multiple UNIQUE constraints can be defined on a table, whereas only one PRIMARY KEY constraint can be defined on a table. Also, unlike PRIMARY KEY constraints, the UNIQUE constraints allow NULL values.

**DEFAULT Constraint**

The DEFAULT constraint specifies the default value for the columns.

A column default is some value that will be inserted in the column by the database engine when an [INSERT](https://www.tutorialrepublic.com/sql-tutorial/sql-insert-statement.php) statement doesn't explicitly assign a particular value.

The following SQL statement creates a default for the *country* column.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=default-constraint)

CREATE TABLE persons (

id INT NOT NULL PRIMARY KEY,

name VARCHAR(30) NOT NULL,

birth\_date DATE,

phone VARCHAR(15) NOT NULL UNIQUE,

country VARCHAR(30) NOT NULL DEFAULT 'Australia'

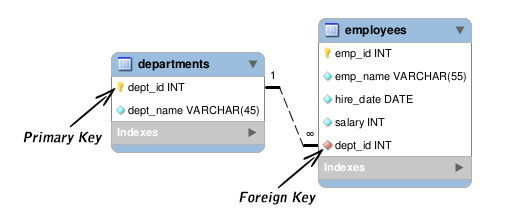
);

**Note:** If you define a table column as NOT NULL, but assign the column a default value, then in the INSERT statement you don't need to explicitly assign a value for that column in order to insert a new row in the table.

**FOREIGN KEY Constraint**

A foreign key (FK) is a column or combination of columns that is used to establish and enforce a relationship between the data in two tables.

Here's a sample diagram showing the relationship between the **employees** and **departments** table. If you look at it carefully, you will notice that the **dept\_id** column of the **employees** table matches the primary key column of the **departments** table. Therefore, the **dept\_id** column of the **employees** table is the foreign key to the **departments** table.



In MySQL you can create a foreign key by defining a FOREIGN KEY constraint when you create a table as follow. The following statement establishes a foreign key on the *dept\_id* column of the *employees* table that references the *dept\_id* column of the *departments* table.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=foreign-key-constraint)

CREATE TABLE employees (

emp\_id INT NOT NULL PRIMARY KEY,

emp\_name VARCHAR(55) NOT NULL,

hire\_date DATE NOT NULL,

salary INT,

dept\_id INT,

FOREIGN KEY (dept\_id) REFERENCES departments(dept\_id)

);

**CHECK Constraint**

The CHECK constraint is used to restrict the values that can be placed in a column.

For example, the range of values for a salary column can be limited by creating a CHECK constraint that allows values only from 3,000 to 10,000. This prevents salaries from being entered beyond the regular salary range. Here's an example:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=check-constraint)

CREATE TABLE employees (

emp\_id INT NOT NULL PRIMARY KEY,

emp\_name VARCHAR(55) NOT NULL,

hire\_date DATE NOT NULL,

salary INT NOT NULL CHECK (salary >= 3000 AND salary <= 10000),

dept\_id INT,

FOREIGN KEY (dept\_id) REFERENCES departments(dept\_id)

);

**Note:** MySQL does not support SQL check constraint. The CHECK clause is parsed however but ignored by all storage engines of the MySQL.

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-create-table-statement.php)

**SQL INSERT Statement**

In this tutorial you will learn how to insert records in a database table using SQL.

**Inserting Data in Table**

In the [preceding chapter](https://www.tutorialrepublic.com/sql-tutorial/sql-create-table-statement.php) we've created a table named *persons* in our *demo* database. Now it's time to insert some data inside our newly created database table.

The INSERT INTO statement is used to insert new rows in a database table.

**Syntax**

The basic syntax for inserting data into a table can be given with:

INSERT INTO *table\_name* (column1,column2,...) VALUES (value1,value2,...);

Here the *column1*, *column2*,..., etc. represents the name of the table columns, whereas the *value1*, *value2*,..., and so on represents the corresponding values for these columns.

Let's insert some records into the *persons* table.

**Step 1: View Table Structure**

Before adding record it's a good idea to obtain the information about the table structure. Execute the following command on MySQL command-line. It will display the information about the columns in the *persons* table i.e. column name, data type, constraints etc.

mysql> DESCRIBE persons;

You can see the column information or structure of any table in MySQL and Oracle database using the command DESCRIBE *table\_name*;, whereas EXEC sp\_columns *table\_name*; in SQL Server (replace the *table\_name* with actual table name).

**Step 2: Adding Records to a Table**

The following statement inserts a new row in *persons* table.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=insert-query)

INSERT INTO persons (name, birth\_date, phone)

VALUES ('Peter Wilson', '1990-07-15', '0711-020361');

Did you notice, we didn't insert any value for id field? Because, if you remember from the [create table](https://www.tutorialrepublic.com/sql-tutorial/sql-create-table-statement.php) chapter, the id field was marked with AUTO\_INCREMENT flag, which tells MySQL to automatically assign a value to this field if it is left unspecified.

**Note:** Non-numeric values like strings and dates must always be surrounded by quotes, whereas numeric values should never be enclosed within quotes. Also, if your string itself contains quotes you should escape it with backslash like 'Let\'s go'.

Similarly, insert another row into the *persons* table, as follow:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=another-insert-query)

INSERT INTO persons (name, birth\_date, phone)

VALUES ('Carrie Simpson', '1995-05-01', '0251-031259');

Insert one more row into the *persons* table, in a similar manner:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=one-more-insert-query)

INSERT INTO persons (name, birth\_date, phone)

VALUES ('Victoria Ashworth', '1996-10-17', '0695-346721');

Now if you select the records from *persons* table, the output will now look like this:

+----+--------------------+------------+-------------+

| id | name | birth\_date | phone |

+----+--------------------+------------+-------------+

| 1 | Peter Wilson | 1990-07-15 | 0711-020361 |

| 2 | Carrie Simpson | 1995-05-01 | 0251-031259 |

| 3 | Victoria Ashworth | 1996-10-17 | 0695-346721 |

+----+--------------------+------------+-------------+

We'll learn about SQL statement for [selecting records from the tables](https://www.tutorialrepublic.com/sql-tutorial/sql-select-statement.php), in the next chapter.

**SQL SELECT Statement**

In this tutorial you will learn how to select records from database tables using SQL.

**Selecting Data from Table**

In the previous chapter we've learned how to insert data in a database table. Now it's time to select the data from existing tables using the SQL query.

The SELECT statement is used to select or retrieve the data from one or more tables. You can use this statement to retrieve all the rows from a table in one go, as well as to retrieve only those rows that satisfy a certain condition or a combination of conditions.

**Syntax**

The basic syntax for selecting the data from a table can be given with:

SELECT *column1\_name*, *column2\_name*, *columnN\_name* FROM *table\_name*;

Here, *column1\_name*, *column2\_name*, ... are the names of the columns or fields of a database table whose values you want to fetch. However, if you want to fetch the values of all the columns available in a table, you can just use the following syntax:

SELECT \* FROM *table\_name*;

Let's put these statements into real use. Suppose we've a table named *employees* in our database that contains the following records:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

+--------+--------------+------------+--------+---------+

**Select All from Table**

The following statement will return all the rows from the *employees* table.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=select-all)

SELECT \* FROM employees;

After execution, the output will look something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

+--------+--------------+------------+--------+---------+

As you can see, it returns all the rows and columns from the *employees* table.

**Tip:** The asterisk (\*) is a wildcard character that means everything. For example, the asterisk character in the SELECT statement of the example above is a shorthand substitute for all the columns of the *employees* table.

**Select Columns from Table**

If you don't require all the data, you can select specific columns, like this:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=select-specific-columns)

SELECT emp\_id, emp\_name, hire\_date, salary

FROM employees;

After executing the above statement, you'll get the output something like this:

+--------+--------------+------------+--------+

| emp\_id | emp\_name | hire\_date | salary |

+--------+--------------+------------+--------+

| 1 | Ethan Hunt | 1995-10-30 | 5000 |

| 2 | Tony Montana | 1990-07-15 | 6500 |

| 3 | Sarah Connor | 2011-04-13 | 5600 |

| 4 | Rick Deckard | 2005-10-18 | 7200 |

| 5 | Martin Blank | 1996-05-24 | 8000 |

+--------+--------------+------------+--------+

As you can see this time there is no *dept\_id* column in the result set. In the next chapter we'll learn how to select the records from a table based on a condition.

**SQL WHERE Clause**

In this tutorial you will learn how to select specific records from a table using SQL.

**Selecting Record Based on Condition**

In the previous chapter we've learnt how to fetch all the records from a table or table columns. But, in real world scenario we generally need to select, update or delete only those records which fulfill certain condition like users who belongs to a certain age group, or country, etc.

The WHERE clause is used with the [SELECT](https://www.tutorialrepublic.com/sql-tutorial/sql-select-statement.php), [UPDATE](https://www.tutorialrepublic.com/sql-tutorial/sql-update-statement.php), and [DELETE](https://www.tutorialrepublic.com/sql-tutorial/sql-delete-statement.php). However, you'll see the use of this clause with other statements in upcoming chapters.

**Syntax**

The WHERE clause is used with the SELECT statement to extract only those records that fulfill specified conditions. The basic syntax can be given with:

SELECT *column\_list* FROM *table\_name* WHERE *condition*;

Here, *column\_list* are the names of columns/fields like *name*, *age*, *country* etc. of a database table whose values you want to fetch. However, if you want to fetch the values of all the columns available in a table, you can use the following syntax:

SELECT \* FROM *table\_name* WHERE *condition*;

Now, let's check out some examples that demonstrate how it actually works.

Suppose we've a table called *employees* in our database with the following records:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

+--------+--------------+------------+--------+---------+

**Filter Records with WHERE Clause**

The following SQL statement will returns all the employees from the *employees* table, whose salary is greater than 7000. The WHERE clause simply filtered out the unwanted data.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=where-clause)

SELECT \* FROM employees

WHERE salary > 7000;

After execution, the output will look something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

+--------+--------------+------------+--------+---------+

As you can see the output contains only those employees whose salary is greater than 7000. Similarly, you can fetch records from specific columns, like this:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=filter-records-with-where-clause)

SELECT emp\_id, emp\_name, hire\_date, salary

FROM employees

WHERE salary > 7000;

After executing the above statement, you'll get the output something like this:

+--------+--------------+------------+--------+

| emp\_id | emp\_name | hire\_date | salary |

+--------+--------------+------------+--------+

| 3 | Sarah Connor | 2005-10-18 | 8000 |

| 4 | Rick Deckard | 2007-01-03 | 7200 |

+--------+--------------+------------+--------+

The following statement will fetch the records of an employee whose employee id is 2.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=select-specific-rows-based-on-condition)

SELECT \* FROM employees

WHERE emp\_id = 2;

This statement will produce the following output:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

+--------+--------------+------------+--------+---------+

This time we got only one row in the output, because *emp\_id* is unique for every employee.

**Operators Allowed in WHERE Clause**

SQL supports a number of different operators that can be used in WHERE clause, the most important ones are summarized in the following table.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| = | Equal | WHERE id = 2 |
| > | Greater than | WHERE age > 30 |
| < | Less than | WHERE age < 18 |
| >= | Greater than or equal | WHERE rating >= 4 |
| <= | Less than or equal | WHERE price <= 100 |
| LIKE | Simple pattern matching | WHERE name LIKE 'Dav' |
| IN | Check whether a specified value matches any value in a list or subquery | WHERE country IN ('USA', 'UK') |
| BETWEEN | Check whether a specified value is within a range of values |  |

**SQL AND & OR Operators**

In this tutorial you will learn how to use the AND & OR operators with the WHERE clause to filter records based on more than one condition.

**Selecting Record Based on Condition**

In the previous chapter we've learned how to fetch records from a table using a single condition with the WHERE clause. But sometimes you need to filter records based on multiple conditions like selecting users whose ages are greater than 30 and country is United States, selecting products whose price is lower than 100 dollar and ratings is greater than 4, etc.

**The AND Operator**

The AND operator is a logical operator that combines two conditions and returns TRUE only if both condition evaluate to TRUE . The AND operator is often used in the [WHERE](https://www.tutorialrepublic.com/sql-tutorial/sql-where-clause.php) clause of the [SELECT](https://www.tutorialrepublic.com/sql-tutorial/sql-select-statement.php), [UPDATE](https://www.tutorialrepublic.com/sql-tutorial/sql-update-statement.php), [DELETE](https://www.tutorialrepublic.com/sql-tutorial/sql-delete-statement.php) statement to form conditions to filter the result set.

SELECT *column1\_name*, *column2\_name*, *columnN\_name*

FROM *table\_name*

WHERE *condition1 AND condition2;*

Let's check out some examples that demonstrate how it actually works.

Suppose we've a table called *employees* in our database with the following records:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

+--------+--------------+------------+--------+---------+

**Using WHERE Clause with AND Operator**

The following SQL statement will return only those employees from the *employees* table whose salary is greater than 7000 and the *dept\_id* is equal to 5.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=and-operator)

SELECT \* FROM employees

WHERE salary > 7000 AND dept\_id = 5;

After execution, you will get the output something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

+--------+--------------+------------+--------+---------+

**The OR Operator**

Similarly, the OR operator is also a logical operator that combines two conditions, but it returns TRUE when either of the conditions is TRUE.

The following SQL statement will return all the employees from the *employees* table whose salary is either greater than 7000 or the *dept\_id* is equal to 5.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=or-operator)

SELECT \* FROM employees

WHERE salary > 7000 OR dept\_id = 5;

This time you will get the output something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

+--------+--------------+------------+--------+---------+

**Combining AND & OR Operator**

You can also combine AND and OR to create complex conditional expressions.

The following SQL statement will return all the employees whose salary is greater than 5000 and the *dept\_id* is either equal to 1 or 5.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=combine-and-or-operators)

SELECT \* FROM employees

WHERE salary > 5000 AND (dept\_id = 1 OR dept\_id = 5);

After executing the above query, you will get the output something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

+--------+--------------+------------+--------+---------+

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-where-clause.php)

**SQL IN & BETWEEN Operators**

In this tutorial you will learn how to use IN and BETWEEN operators with WHERE clause.

**Working with Range and Membership Conditions**

In the previous chapter we've learned how to combine multiple conditions using the AND and OR operators. However, sometimes this is not sufficient and very productive, for example, if you have to check the values that lie within a range or set of values.

And here the IN and BETWEEN operators comes in picture that lets you define an exclusive range or a set of values rather than combining the separate conditions.

**The IN Operator**

The IN operator is logical operator that is used to check whether a particular value exists within a set of values or not. Its basic syntax can be given with:

SELECT *column\_list* FROM *table\_name*  
WHERE *column\_name* IN (*value1*, *value1*,...);

Here, *column\_list* are the names of columns/fields like *name*, *age*, *country* etc. of a database table whose values you want to fetch. Well, let's check out some examples.

Consider we've an *employees* table in our database that has following records:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

+--------+--------------+------------+--------+---------+

The following SQL statement will return only those employees whose *dept\_id* is either 1 or 3.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=in-operator)

SELECT \* FROM employees

WHERE dept\_id IN (1, 3);

After executing the query, you will get the result set something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

+--------+--------------+------------+--------+---------+

Similarly, you can use the NOT IN operator, which is exact opposite of the IN. The following SQL statement will return all the employees except those whose *dept\_id* is not 1 or 3.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=not-in-operator)

SELECT \* FROM employees

WHERE dept\_id NOT IN (1, 3);

After executing the query, this time you will get the result set something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

+--------+--------------+------------+--------+---------+

**The BETWEEN Operator**

Sometimes you want to select a row if the value in a column falls within a certain range. This type of condition is common when working with numeric data.

To perform the query based on such condition you can utilize the BETWEEN operator. It is a logical operator that allows you to specify a range to test, as follow:

SELECT *column1\_name*, *column2\_name*, *columnN\_name*

FROM *table\_name*

WHERE *column\_name* BETWEEN *min\_value* AND *max\_value*;

Let's build and perform the queries based upon range conditions on our *employees* table.

**Define Numeric Ranges**

The following SQL statement will return only those employees from the *employees* table, whose salary falls within the range of 7000 and 9000.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=define-a-numeric-range-with-between-operator)

SELECT \* FROM employees

WHERE salary BETWEEN 7000 AND 9000;

After execution, you will get the output something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

+--------+--------------+------------+--------+---------+

**Define Date Ranges**

When using the BETWEEN operator with date or time values, use the CAST() function to explicitly convert the values to the desired data type for best results. For example, if you use a string such as '2016-12-31' in a comparison to a DATE, cast the string to a DATE, as follow:

The following SQL statement selects all the employees who hired between 1st January 2006 (i.e. '2006-01-01') and 31st December 2016 (i.e. '2016-12-31'):

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=define-a-date-range-with-between-operator)

SELECT \* FROM employees WHERE hire\_date

BETWEEN CAST('2006-01-01' AS DATE) AND CAST('2016-12-31' AS DATE);

After executing the query, you will get the result set something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

+--------+--------------+------------+--------+---------+

**Define String Ranges**

While ranges of dates and numbers are most common, you can also build conditions that search for ranges of strings. The following SQL statement selects all the employees whose name beginning with any of the letter between 'O' and 'Z':

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=define-a-string-range-with-between-operator)

SELECT \* FROM employees

WHERE emp\_name BETWEEN 'O' AND 'Z';

After execution, you will get the output something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

+--------+--------------+------------+--------+---------+

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-and-or-operators.php)[NEXT PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-order-by-clause.php)

**SQL ORDER BY Clause**

In this tutorial you will learn how to sort data returned by a SELECT query in SQL.

**Ordering the Result Set**

Generally when you use the SELECT statement to fetch data from a table, the rows in result set are not in any particular order. If you want your result set in a particular order, you can specify the ORDER BY clause at the end of the statement which tells the server how to sort the data returned by the query. The default sorting order is ascending.

**Syntax**

The ORDER BY clause is used to sort the data returned by a query in ascending or descending order. The basic syntax of this clause can be given with:

SELECT *column\_list* FROM *table\_name* ORDER BY column\_name ASC|DESC;

Here, *column\_list* are the names of columns/fields like *name*, *age*, *country* etc. of a database table whose values you want to fetch, whereas the *column\_name* is name of the column you want to sort. Let's check out some examples that demonstrate how it actually works.

Consider we've an *employees* table in our database that has following records:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

+--------+--------------+------------+--------+---------+

**Sorting Single Column**

The following SQL statement will return all the employees from the *employees* table and orders the result set by the *emp\_name* column in ascending order.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=sort-results-in-ascending-order)

SELECT \* FROM employees

ORDER BY emp\_name ASC;

You can skip the ASC option and simply use the following syntax. It returns the same result set as previous statement, because the default sorting order is ascending:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=default-sort-order)

SELECT \* FROM employees

ORDER BY emp\_name;

After executing the above command, you'll get the output something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

+--------+--------------+------------+--------+---------+

Similarly, you can use the DESC option to perform a sorting in descending order. The following statement will orders the result set by the numeric *salary* column in descending order.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=sort-results-in-descending-order)

SELECT \* FROM employees

ORDER BY salary DESC;

This time, you'll get the result set something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

+--------+--------------+------------+--------+---------+

**Sorting Multiple Columns**

You can also specify multiple columns while sorting. However, the change in result set will not visible until you've some duplicate values in your table. Well, let's find out:

To understand the multi-column sorting in a better way, let's assume that we've a table named *trainees* in our database with the following records:

+----+------------+------------+-------------+--------+

| id | first\_name | last\_name | birth\_date | gender |

+----+------------+------------+-------------+--------+

| 1 | Peter | Parker | 1998-03-04 | M |

| 2 | Harry | Potter | 2001-08-30 | M |

| 3 | Peter | Pan | 2004-09-19 | M |

| 4 | Alice | Kingsleigh | 1999-07-02 | F |

| 5 | John | Connor | 2002-01-15 | M |

+----+------------+------------+-------------+--------+

If you see the table carefully, you'll find that we've some duplicate values. However, the full name of the trainee "Peter Parker" and "Peter Pan" are different but their first names are same.

Now execute the following command which orders the result set by the *first\_name* column.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=sort-on-a-single-column)

SELECT \* FROM trainees

ORDER BY first\_name;

After execution, you'll get the output something like this:

+----+------------+------------+-------------+--------+

| id | first\_name | last\_name | birth\_date | gender |

+----+------------+------------+-------------+--------+

| 4 | Alice | Kingsleigh | 1999-07-02 | F |

| 2 | Harry | Potter | 2001-08-30 | M |

| 5 | John | Connor | 2002-01-15 | M |

| 1 | Peter | Parker | 1998-03-04 | M |

| 3 | Peter | Pan | 2004-09-19 | M |

+----+------------+------------+-------------+--------+

Now execute this statement which orders the result set by *first\_name* and *last\_name* columns.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=sort-on-more-than-one-column)

SELECT \* FROM trainees

ORDER BY first\_name, last\_name;

+----+------------+------------+-------------+--------+

| id | first\_name | last\_name | birth\_date | gender |

+----+------------+------------+-------------+--------+

| 4 | Alice | Kingsleigh | 1999-07-02 | F |

# | 2 | Harry **SQL TOP / MySQL LIMIT Clause**

In this tutorial you will learn how to retrieve fixed number of records from the table.

**Limiting Result Sets**

In some situations, you may not be interested in all of the rows returned by a query, for example, if you just want to retrieve the top 10 employees who recently joined the organization, get top 3 students by score, or something like that.

To handle such situations, you can use SQL's TOP clause in your SELECT statement. However the TOP clause is only supported by the SQL Server and MS Access database systems.

MySQL provides an equivalent LIMIT clause, whereas Oracle provides ROWNUM clause for the SELECT statement to restrict the number of rows returned by a query.

**SQL TOP Syntax**

The SQL TOP clause is used to limit the number of rows returned. Its basic syntax is:

SELECT TOP *number* | *percent* *column\_list* FROM *table\_name*;

Here, *column\_list* is a comma separated list of column or field names of a database table (e.g. *name*, *age*, *country*, etc.) whose values you want to fetch. Let's see how it works.

Suppose we've an *employees* table in our database with the following records:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

+--------+--------------+------------+--------+---------+

The following statement returns top three highest-paid employees from the *employees* table.

**Example**

[**Try this code »**](javascript:void(0);)

-- Syntax for SQL Server Database

SELECT TOP 3 \* FROM employees

ORDER BY salary DESC;

The result set returned will look something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

+--------+--------------+------------+--------+---------+

You can optionally use the PERCENT keyword after the fixed value in a TOP clause, if you just want to retrieve the percentage of rows instead of fixed number of rows. Fractional values are rounded up to the next integer value (e.g. 1.5 rounded to 2).

The following statement returns top 30 percent of the highest-paid employees.

**Example**

[**Try this code »**](javascript:void(0);)

-- Syntax for SQL Server Database

SELECT TOP 30 PERCENT \* FROM employees

ORDER BY salary DESC;

The result set returned by the above query will look like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

+--------+--------------+------------+--------+---------+

**MySQL LIMIT Syntax**

The MySQL's LIMIT clause does the same work as SQL TOP clause. Its basic syntax is:

SELECT *column\_list* FROM *table\_name* LIMIT *number*;

The following statement returns top three highest-paid employees from the *employees* table.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=limit-clause)

-- Syntax for MySQL Database

SELECT \* FROM employees

ORDER BY salary DESC LIMIT 3;

After execution, you'll get the output something like this:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

+--------+--------------+------------+--------+---------+

**Note:** In a SELECT statement, always use an [ORDER BY](https://www.tutorialrepublic.com/sql-tutorial/sql-order-by-clause.php) clause with the LIMIT clause. Otherwise, you may not get the desired result.

**Setting Row Offset in LIMIT Clause**

The LIMIT clause accepts an optional second parameter.

When two parameters are specified, the first parameter specifies the offset of the first row to return i.e. the starting point, whereas the second parameter specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1).

So, if you want to find out the third-highest paid employee, you can do the following:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=limit-clause-with-two-parameters)

-- Syntax for MySQL Database

SELECT \* FROM employees

ORDER BY salary DESC LIMIT 2, 1;

After executing the above command, you'll get only one record in your result set:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

+--------+--------------+------------+--------+---------+

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-order-by-clause.php) | Potter | 2001-08-30 | M |

| 5 | John | Connor | 2002-01-15 | M |

| 3 | Peter | Pan | 2004-09-19 | M |

| 1 | Peter | Parker | 1998-03-04 | M |

+----+------------+------------+-------------+--------+

Did you notice the difference between the previous and the current result set — this time the record of the trainee "Peter Parker" comes after the "Peter Pan".

Since the first name of both the trainees are same which is "Peter", so the second level ordering is performed at the *last\_name* column for these two trainees that's why the record of the trainee "Peter Parker" comes after the "Peter Pan".

**Note:** When multiple sort columns are specified, the result set is initially sorted by the first column and then that ordered list is sorted by the second column, and so on.

**SQL DISTINCT Clause**

In this tutorial you will learn how to remove duplicate values from a result set.

**Retrieving Distinct Values**

When fetching data from a database table, the result set may contain duplicate rows or values. If you want to remove these duplicate values you can specify the keyword DISTINCT directly after the SELECT keyword, as demonstrated below:

**Syntax**

The DISTINCT clause is used to remove duplicate rows from the result set:

SELECT DISTINCT *column\_list* FROM *table\_name*;

Here, *column\_list* is a comma separated list of column or field names of a database table (e.g. *name*, *age*, *country*, etc.) whose values you want to fetch.

**Note:** The DISTINCT clause behaves similar to the [UNIQUE](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php#unique) constraint, except in the way it treats nulls. Two NULL values are considered unique, while at the same time they are not considered distinct from each other.

Let's check out some examples that demonstrate how it actually works.

Suppose we've a *customers* table in our database with the following records:

+---------+--------------------+-----------+-------------+

| cust\_id | cust\_name | city | postal\_code |

+---------+--------------------+-----------+-------------+

| 1 | Maria Anders | Berlin | 12209 |

| 2 | Fran Wilson | Madrid | 28023 |

| 3 | Dominique Perrier | Paris | 75016 |

| 4 | Martin Blank | Turin | 10100 |

| 5 | Thomas Hardy | Portland | 97219 |

| 6 | Christina Aguilera | Madrid | 28001 |

+---------+--------------------+-----------+-------------+

Now execute the following statement which returns all the rows from the *city* column of this table.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=select-all-values-from-a-table-column)

SELECT city FROM customers;

After execution, you'll get the output something like this:

+-----------+

| city |

+-----------+

| Berlin |

| Madrid |

| Paris |

| Turin |

| Portland |

| Madrid |

+-----------+

If you see the output carefully, you'll find the city "Madrid" appears two times in our result set, which is not good. Well, let's fix this problem.

**Removing Duplicate Data**

The following statement uses DISTINCT to generate a list of all city in the *customers* table.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=select-distinct-values-from-a-table-column)

SELECT DISTINCT city FROM customers;

After executing the above command, you'll get the output something like this:

+-----------+

| city |

+-----------+

| Berlin |

| Madrid |

| Paris |

| Turin |

| Portland |

+-----------+

As you see this time there is no duplicate values in our result set.

**Note:** If you use the SELECT DISTINCT statement for a column that has multiple NULL values, Then SQL keeps one NULL value and removes others from the result set, because DISTINCT treats all the NULL values as the same value.

# [PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-top-clause.php)**SQL DELETE Statement**

In this tutorial you will learn how to delete records from a database table using SQL.

**Deleting Data from Tables**

Just as you insert records into a table with the [INSERT](https://www.tutorialrepublic.com/sql-tutorial/sql-insert-statement.php) statement, you can delete records from a table as well with the DELETE statement.

**Syntax**

The DELETE statement is used to remove one or more rows from a table.

DELETE FROM *table\_name* WHERE *condition*;

**Warning:** The [WHERE](https://www.tutorialrepublic.com/sql-tutorial/sql-where-clause.php) clause in the DELETE statement specifies which record or records should be deleted. It is however optional, but if you omit or forget the WHERE clause, all the records will be deleted permanently from the table.

Let's delete some records from the *persons* table that we've created in [create table](https://www.tutorialrepublic.com/sql-tutorial/sql-create-table-statement.php) chapter.

Suppose that our *persons* table currently has the following records:

+----+--------------------+------------+-------------+

| id | name | birth\_date | phone |

+----+--------------------+------------+-------------+

| 1 | Peter Wilson | 1990-07-15 | 0711-020361 |

| 2 | Carrie Simpson | 1995-05-01 | 0251-031259 |

| 3 | Victoria Ashworth | 1996-10-17 | 0695-346721 |

| 4 | George Bailey | 1993-03-05 | 0897-034214 |

| 5 | Norman Bates | 1999-08-25 | 0522-556721 |

+----+--------------------+------------+-------------+

**Delete Records Based on Conditions**

The following statement will delete the rows from *persons* table where *id* is greater than 3.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=delete-specific-rows)

DELETE FROM persons WHERE id > 3;

After executing the query, the *persons* table will look something like this:

+----+--------------------+------------+-------------+

| id | name | birth\_date | phone |

+----+--------------------+------------+-------------+

| 1 | Peter Wilson | 1990-07-15 | 0711-020361 |

| 2 | Carrie Simpson | 1995-05-01 | 0251-031259 |

| 3 | Victoria Ashworth | 1996-10-17 | 0695-346721 |

+----+--------------------+------------+-------------+

**Note:** See the tutorial on [SQL WHERE clause](https://www.tutorialrepublic.com/sql-tutorial/sql-where-clause.php) to know how to form complex query based on multiple conditions while deleting the records from the tables.

**Delete All Data**

Similarly, as mentioned above if you do not specify the WHERE clause in the DELETE statement all the rows from the table will be deleted. However, the target table itself won't be deleted that means the table structure, attributes, and indexes will remain intact.

The following statement will remove all the records from the *persons* table:

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=delete-all-rows)

DELETE FROM persons;

Now, if you try to [select the records](https://www.tutorialrepublic.com/sql-tutorial/sql-select-statement.php) from the *persons* table you'll get an empty result set.

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-update-statement.php)

**SQL TRUNCATE TABLE Statement**

In this tutorial you will learn how to quickly delete all rows from a table using SQL.

**Removing Table Data**

The TRUNCATE TABLE statement removes all the rows from a table more quickly than a DELETE. Logically, TRUNCATE TABLE is similar to the [DELETE](https://www.tutorialrepublic.com/sql-tutorial/sql-delete-statement.php) statement with no [WHERE](https://www.tutorialrepublic.com/sql-tutorial/sql-where-clause.php) clause.

The TRUNCATE TABLE statement removes all the rows from a table, but the table structure and its columns, constraints, indexes, and so on remain intact. To remove the table definition in addition to its data, you can use the [DROP TABLE](https://www.tutorialrepublic.com/sql-tutorial/sql-drop-statement.php) statement.

**Syntax**

The basic syntax of TRUNCATE TABLE can be given with:

TRUNCATE TABLE *table\_name*;

Let's perform the truncate operation on a database table.

Consider we've an *employees* table in our database with the following records:

+--------+--------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+--------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

+--------+--------------+------------+--------+---------+

The following command removes all the rows from the *employees* table:

**Example**

[**Try this code »**](javascript:void(0);)

TRUNCATE TABLE employees;

Now, after executing the above SQL statement, if you try to [select the records](https://www.tutorialrepublic.com/sql-tutorial/sql-select-statement.php) from the *employees* table, you will get an empty result set.

**TRUNCATE TABLE vs DELETE**

Although DELETE and TRUNCATE TABLE seem to have the same effect, but they do work differently. Here are some major differences between these two statements:

* TRUNCATE TABLE statement drop and re-create the table in such a way that any  
  auto-increment value is reset to its start value which is generally 1.
* DELETE lets you filter which rows to be deleted based upon an optional WHERE clause, whereas TRUNCATE TABLE doesn't support WHERE clause it just removes all the rows.
* TRUNCATE TABLE is faster and uses fewer system resources than DELETE, because DELETE scans the table to generate a count of rows that were affected then delete the rows one by one and records an entry in the database log for each deleted row, while TRUNCATE TABLE just delete all the rows without providing any additional information.

**Tip:** Use TRUNCATE TABLE if you just want to delete all the rows and re-create the whole table. Use DELETE either if you want to delete limited number of rows based on specific condition or you don't want to reset the auto-increment value.

**SQL DROP Statement**

In this tutorial you will learn how to delete database and table using SQL.

**Removing a Table from Database**

You can use the DROP TABLE statement to easily delete the database tables that you no longer need. The DROP TABLE statement permanently erase all data from the table, as well as the metadata that defines the table in the data dictionary.

**Syntax**

The DROP TABLE removes one or more tables. The syntax can be given with:

DROP TABLE *table1\_name*, *table2\_name*, ...;

Here, *table1\_name*, *table2\_name*, ... are the names of the tables that you want to delete.

**Warning:** Dropping a database or table is irreversible. So, be careful while using the DROP statement, because database system generally do not display any alert like "Are you sure?". It will immediately delete the database or table and all of its data.

Let's try to remove a database table using the DROP TABLE statement.

If you remember back to [create table](https://www.tutorialrepublic.com/sql-tutorial/sql-create-table-statement.php) chapter, we've created a table *persons* in our *demo* database. The following statement will remove this table permanently from the database.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=delete-table-from-the-database)

DROP TABLE persons;

After executing the above command, if you try to perform any operation on the *persons* table, like selecting the records from it, you'll get an error message.

**Removing Database**

Similarly, you can delete a database using the DROP DATABASE statement. The following command will permanently remove the *demo* database from the database server.

**Example**

[**Try this code »**](javascript:void(0);)

DROP DATABASE demo;

Now if you try to select the *demo* database using the USE demo; statement, you'll get an error message saying "Unknown database" or "Database does not exist".

**SQL Joining Tables**

In this tutorial you will learn how to join two tables to get combined data.

**SQL Join Fundamentals**

All the queries you've seen so far have been concentrated on a single table. But in real life situation you often need to query two or more tables at time and bring a combined result set. This is technically referred to as a join, since it involves joining different tables, based on a common field between them (the [foreign key](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php#foreign-key)) to create new views of the data.

To understand this easily, let's look at the following *employees* and *departments* tables. Here, the *dept\_id* column of the *employees* table is the foreign key to the *departments* table. Therefore, these two tables can be joined to get the combined data.

|  |  |  |
| --- | --- | --- |
| +--------+--------------+------------+---------+  | emp\_id | emp\_name | hire\_date | dept\_id |  +--------+--------------+------------+---------+  | 1 | Ethan Hunt | 2001-05-01 | 4 |  | 2 | Tony Montana | 2002-07-15 | 1 |  | 3 | Sarah Connor | 2005-10-18 | 5 |  | 4 | Rick Deckard | 2007-01-03 | 3 |  | 5 | Martin Blank | 2008-06-24 | NULL |  +--------+--------------+------------+---------+ |  | +---------+------------------+  | dept\_id | dept\_name |  +---------+------------------+  | 1 | Administration |  | 2 | Customer Service |  | 3 | Finance |  | 4 | Human Resources |  | 5 | Sales |  +---------+------------------+ |
| Table: **employees** |  | Table: **departments** |

**Note:** In order to join tables, data of the columns which are used for joining tables should match, not necessarily the column names.

**Types of Joins**

When you join tables, the type of join that you create in your query affects the rows that appear in the result set. You can create the following types of joins:

**Inner join**

A join that returns only those rows that have a match in both joined tables. For example, you can join the *employees* and *departments* tables to create a result set that shows the department name for each employee. In an inner join, employees for which there is no department information are not included in the result set, nor are departments with no employees.

We will learn more about [inner join](https://www.tutorialrepublic.com/sql-tutorial/sql-inner-join-operation.php) in next chapter.

**Outer join**

Outer joins are an extension to inner joins. An outer join returns the rows even if they don't have related rows in the joined table. There are three types of outer joins: left outer join (*or* [left join](https://www.tutorialrepublic.com/sql-tutorial/sql-left-join-operation.php)), right outer join (*or* [right join](https://www.tutorialrepublic.com/sql-tutorial/sql-right-join-operation.php)), and full outer join (*or* [full join](https://www.tutorialrepublic.com/sql-tutorial/sql-full-join-operation.php)).

We will learn more about these variations of the outer join in later chapters.

**Cross join**

Cross joins are joins without a join condition. Each row of one table is combined with each row of another table. This type of result set is called a Cartesian product or cross product. For example, a cross join between the *employees* and *departments* tables yields a result set with one row for each possible employees/departments combination.

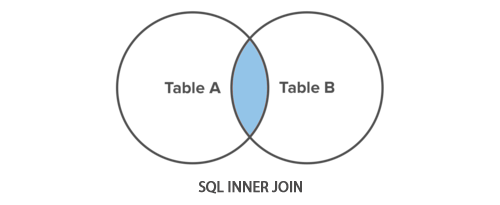
We will learn more about [cross join](https://www.tutorialrepublic.com/sql-tutorial/sql-cross-join-operation.php) in upcoming chapters.

**SQL INNER JOIN Operation**

In this tutorial you will learn how to fetch data from two tables using SQL inner join.

**Using Inner Joins**

The INNER JOIN is the most common [type of join](https://www.tutorialrepublic.com/sql-tutorial/sql-joining-tables.php#types-of-joins). It returns only those rows that have a match in both joined tables. The following Venn diagram illustrates how inner join works.



To understand this easily, let's look at the following *employees* and *departments* tables.

|  |  |  |
| --- | --- | --- |
| +--------+--------------+------------+---------+  | emp\_id | emp\_name | hire\_date | dept\_id |  +--------+--------------+------------+---------+  | 1 | Ethan Hunt | 2001-05-01 | 4 |  | 2 | Tony Montana | 2002-07-15 | 1 |  | 3 | Sarah Connor | 2005-10-18 | 5 |  | 4 | Rick Deckard | 2007-01-03 | 3 |  | 5 | Martin Blank | 2008-06-24 | NULL |  +--------+--------------+------------+---------+ |  | +---------+------------------+  | dept\_id | dept\_name |  +---------+------------------+  | 1 | Administration |  | 2 | Customer Service |  | 3 | Finance |  | 4 | Human Resources |  | 5 | Sales |  +---------+------------------+ |
| Table: **employees** |  | Table: **departments** |

Now, let's say you need to retrieve the id, name, hire date, and the department name of only those employees who assigned to a particular department. Because, in real-life scenario there may be some employees who are not yet assigned to a department, like the fifth employee "Martin Blank" in our *employees* table. But the question here is, how to retrieve the data from both the tables in the same SQL query? Well, let's find out.

If you see the *employees* table, you'll notice that it has a column named *dept\_id* which holds the ID of the department to which each employee is assigned i.e. in technical terms, the *employees* table's *dept\_id* column is the foreign key to the *departments* table, and therefore we will use this column as a bridge between these two tables.

Here's an example that retrieves the employee's id, name, hiring date and their department by joining the *employees* and *departments* tables together using the common *dept\_id* column. It excludes those employees who are not assigned to any department.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=inner-join)

SELECT t1.emp\_id, t1.emp\_name, t1.hire\_date, t2.dept\_name

FROM employees AS t1 INNER JOIN departments AS t2

ON t1.dept\_id = t2.dept\_id ORDER BY emp\_id;

**Tip:** When joining tables, prefix each column name with the name of the table it belongs to (e.g. employees.dept\_id, departments.dept\_id, or t1.dept\_id, t2.dept\_id if you're using the [table aliases](https://www.tutorialrepublic.com/sql-tutorial/sql-aliases.php)) in order to avoid confusion and ambiguous column error in case columns in different tables have the same name.

**Note:** To save time, in place of typing the long table names you can use [table aliases](https://www.tutorialrepublic.com/sql-tutorial/sql-aliases.php) in the query. For example, you can give the *employees* table an alias name t1 and refer its column emp\_name using t1.emp\_name instead of employees.emp\_name

After executing the above command, you get the result set something like this:

+--------+--------------+------------+-----------------+

| emp\_id | emp\_name | hire\_date | dept\_name |

+--------+--------------+------------+-----------------+

| 1 | Ethan Hunt | 2001-05-01 | Human Resources |

| 2 | Tony Montana | 2002-07-15 | Administration |

| 3 | Sarah Connor | 2005-10-18 | Sales |

| 4 | Rick Deckard | 2007-01-03 | Finance |

+--------+--------------+------------+-----------------+

As you can see, the result set contains only those employees for which the *dept\_id* value is present and that value also exists in the *dept\_id* column of the *departments* table.

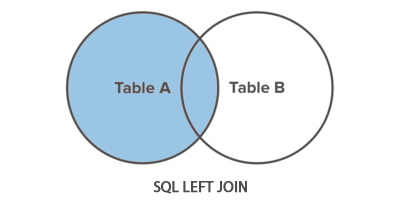
# [PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-joining-tables.php)**SQL LEFT JOIN Operation**

In this tutorial you will learn how to retrieve data from two tables using SQL left join.

**Using Left Joins**

A LEFT JOIN statement returns all rows from the left table along with the rows from the right table for which the join condition is met. Left join is a type of [outer join](https://www.tutorialrepublic.com/sql-tutorial/sql-joining-tables.php#outer-join) that's why it is also referred as *left outer join*. Other variations of outer join are [right join](https://www.tutorialrepublic.com/sql-tutorial/sql-right-join-operation.php) and [full join](https://www.tutorialrepublic.com/sql-tutorial/sql-full-join-operation.php).

The following Venn diagram illustrates how left join works.



**Note:** An outer join is a join that includes rows in a result set even though there may not be a match between rows in the two tables being joined.

To understand this clearly, let's look at the following *employees* and *departments* tables.

|  |  |  |
| --- | --- | --- |
| +--------+--------------+------------+---------+  | emp\_id | emp\_name | hire\_date | dept\_id |  +--------+--------------+------------+---------+  | 1 | Ethan Hunt | 2001-05-01 | 4 |  | 2 | Tony Montana | 2002-07-15 | 1 |  | 3 | Sarah Connor | 2005-10-18 | 5 |  | 4 | Rick Deckard | 2007-01-03 | 3 |  | 5 | Martin Blank | 2008-06-24 | NULL |  +--------+--------------+------------+---------+ |  | +---------+------------------+  | dept\_id | dept\_name |  +---------+------------------+  | 1 | Administration |  | 2 | Customer Service |  | 3 | Finance |  | 4 | Human Resources |  | 5 | Sales |  +---------+------------------+ |
| Table: **employees** |  | Table: **departments** |

Now, let's say you want to retrieve the id, name and hire date of all the employees along with the name of their department, irrespective of whether they are assigned to any department or not. To get such type of result set we need to apply a left join.

The following statement retrieves employee's id, name, hiring date and their department name by joining the *employees* and *departments* tables together using the common *dept\_id* field. It also includes those employees who are not assigned to a department.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=left-join)

SELECT t1.emp\_id, t1.emp\_name, t1.hire\_date, t2.dept\_name

FROM employees AS t1 LEFT JOIN departments AS t2

ON t1.dept\_id = t2.dept\_id ORDER BY emp\_id;

**Tip:** In a join query, the left table is the one that appears leftmost in the JOIN clause, and the right table is the one that appears rightmost.

After executing the above command, you'll get the output something like this:

+--------+--------------+------------+-----------------+

| emp\_id | emp\_name | hire\_date | dept\_name |

+--------+--------------+------------+-----------------+

| 1 | Ethan Hunt | 2001-05-01 | Human Resources |

| 2 | Tony Montana | 2002-07-15 | Administration |

| 3 | Sarah Connor | 2005-10-18 | Sales |

| 4 | Rick Deckard | 2007-01-03 | Finance |

| 5 | Martin Blank | 2008-06-24 | NULL |

+--------+--------------+------------+-----------------+

As you can clearly see the left join includes all the rows from the *employees* table in the result set, whether or not there is a match on the *dept\_id* column in the *departments* table.

**Note:** If there is a row in the left table but no match in the right table, then the associated result row contains NULL values for all columns coming from the right table.

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-inner-join-operation.php)

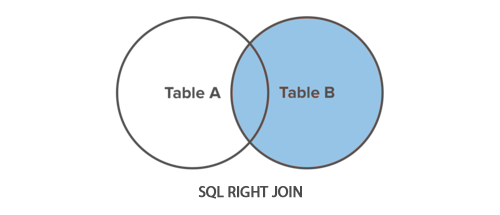
**SQL RIGHT JOIN Operation**

In this tutorial you will learn how to fetch data from two tables using SQL right join.

**Using Right Joins**

The RIGHT JOIN is the exact opposite of the [LEFT JOIN](https://www.tutorialrepublic.com/sql-tutorial/sql-left-join-operation.php). It returns all rows from the right table along with the rows from the left table for which the join condition is met.

Right join is a type of [outer join](https://www.tutorialrepublic.com/sql-tutorial/sql-joining-tables.php#outer-join) that's why it is also referred as *right outer join*. Other variations of outer join are [left join](https://www.tutorialrepublic.com/sql-tutorial/sql-left-join-operation.php) and [full join](https://www.tutorialrepublic.com/sql-tutorial/sql-full-join-operation.php). The following Venn diagram illustrates how right join works.



**Note:** An outer join is a join that includes rows in a result set even though there may not be a match between rows in the two tables being joined.

To understand this clearly, let's look at the following *employees* and *departments* tables.

|  |  |  |
| --- | --- | --- |
| +--------+--------------+------------+---------+  | emp\_id | emp\_name | hire\_date | dept\_id |  +--------+--------------+------------+---------+  | 1 | Ethan Hunt | 2001-05-01 | 4 |  | 2 | Tony Montana | 2002-07-15 | 1 |  | 3 | Sarah Connor | 2005-10-18 | 5 |  | 4 | Rick Deckard | 2007-01-03 | 3 |  | 5 | Martin Blank | 2008-06-24 | NULL |  +--------+--------------+------------+---------+ |  | +---------+------------------+  | dept\_id | dept\_name |  +---------+------------------+  | 1 | Administration |  | 2 | Customer Service |  | 3 | Finance |  | 4 | Human Resources |  | 5 | Sales |  +---------+------------------+ |
| Table: **employees** |  | Table: **departments** |

Now, let's say you want to retrieve the names of all departments as well as the details of employees who're working in that department. But, in real situation there may be some department in which currently no employee is working. Well, let's find out.

The following statement retrieves all the available departments as well as the id, name, hiring date of the employees who belongs to that department by joining the *employees* and *departments* tables together using the common *dept\_id* field.

**Example**

[**Try this code »**](javascript:void(0);)

SELECT t1.emp\_id, t1.emp\_name, t1.hire\_date, t2.dept\_name

FROM employees AS t1 RIGHT JOIN departments AS t2

ON t1.dept\_id = t2.dept\_id ORDER BY dept\_name;

**Tip:** In a join query, the left table is the one that appears leftmost in the JOIN clause, and the right table is the one that appears rightmost.

After executing the above command, you'll get the output something like this:

+--------+--------------+------------+------------------+

| emp\_id | emp\_name | hire\_date | dept\_name |

+--------+--------------+------------+------------------+

| 2 | Tony Montana | 2002-07-15 | Administration |

| NULL | NULL | NULL | Customer Service |

| 4 | Rick Deckard | 2007-01-03 | Finance |

| 1 | Ethan Hunt | 2001-05-01 | Human Resources |

| 3 | Sarah Connor | 2005-10-18 | Sales |

+--------+--------------+------------+------------------+

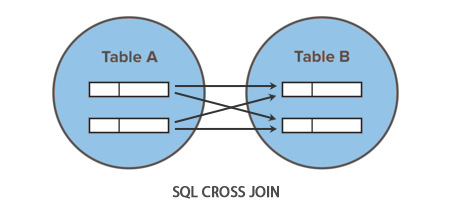
The right join includes all the rows from the *departments* table in the result set, whether or not there is a match on the *dept\_id* column in the *employees* table, as you can clearly see the department "Customer Service" is included even if there is no employee in this department.

# **Note:** If there is a row in the right table but no match in the left table, then the associated result row contains NULL values for all columns coming from the le**SQL CROSS JOIN Operation**

In this tutorial you will learn how to fetch data from two tables using SQL cross join.

**Using Cross Joins**

If you don't specify a join condition when joining two tables, database system combines each row from the first table with each row from the second table. This type of join is called a cross join or a Cartesian product. The following Venn diagram illustrates how cross join works.



To understand this easily, let's look at the following *employees* and *departments* tables.

|  |  |  |
| --- | --- | --- |
| +--------+--------------+------------+---------+  | emp\_id | emp\_name | hire\_date | dept\_id |  +--------+--------------+------------+---------+  | 1 | Ethan Hunt | 2001-05-01 | 4 |  | 2 | Tony Montana | 2002-07-15 | 1 |  | 3 | Sarah Connor | 2005-10-18 | 5 |  | 4 | Rick Deckard | 2007-01-03 | 3 |  | 5 | Martin Blank | 2008-06-24 | NULL |  +--------+--------------+------------+---------+ |  | +---------+------------------+  | dept\_id | dept\_name |  +---------+------------------+  | 1 | Administration |  | 2 | Customer Service |  | 3 | Finance |  | 4 | Human Resources |  | 5 | Sales |  +---------+------------------+ |
| Table: **employees** |  | Table: **departments** |

The number of rows in a cross join is the product of the number of rows in each table. Here's a simple example of a cross join operation.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=cross-join)

SELECT t1.emp\_id, t1.emp\_name, t1.hire\_date, t2.dept\_name

FROM employees AS t1 CROSS JOIN departments AS t2;

**Tip:** A cross join creates a Cartesian product or multiplication of all rows in one table with all rows in another. So, for example, if one table has 5 rows and another has 10 rows, a cross-join query produces 50 rows, the product of 5 and 10.

After executing the above command, you get the result set something like this:

+--------+--------------+------------+------------------+

| emp\_id | emp\_name | hire\_date | dept\_name |

+--------+--------------+------------+------------------+

| 1 | Ethan Hunt | 2001-05-01 | Administration |

| 2 | Tony Montana | 2002-07-15 | Administration |

| 3 | Sarah Connor | 2005-10-18 | Administration |

| 4 | Rick Deckard | 2007-01-03 | Administration |

| 5 | Martin Blank | 2008-06-24 | Administration |

| 1 | Ethan Hunt | 2001-05-01 | Customer Service |

| 2 | Tony Montana | 2002-07-15 | Customer Service |

| 3 | Sarah Connor | 2005-10-18 | Customer Service |

| 4 | Rick Deckard | 2007-01-03 | Customer Service |

| 5 | Martin Blank | 2008-06-24 | Customer Service |

| 1 | Ethan Hunt | 2001-05-01 | Finance |

| 2 | Tony Montana | 2002-07-15 | Finance |

| 3 | Sarah Connor | 2005-10-18 | Finance |

| 4 | Rick Deckard | 2007-01-03 | Finance |

| 5 | Martin Blank | 2008-06-24 | Finance |

| 1 | Ethan Hunt | 2001-05-01 | Human Resources |

| 2 | Tony Montana | 2002-07-15 | Human Resources |

| 3 | Sarah Connor | 2005-10-18 | Human Resources |

| 4 | Rick Deckard | 2007-01-03 | Human Resources |

| 5 | Martin Blank | 2008-06-24 | Human Resources |

| 1 | Ethan Hunt | 2001-05-01 | Sales |

| 2 | Tony Montana | 2002-07-15 | Sales |

| 3 | Sarah Connor | 2005-10-18 | Sales |

| 4 | Rick Deckard | 2007-01-03 | Sales |

| 5 | Martin Blank | 2008-06-24 | Sales |

+--------+--------------+------------+------------------+

As you can see a cross join is not as useful as the other joins that we've covered in the previous chapters. Since the query didn't specify a join condition, each row from the employees table is combined with each row from the departments table. Therefore, unless you are sure that you want a Cartesian product don't use a cross join.

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-full-join-operation.php)[NEXT PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-union-operation.php)

ADVERTISEMENTS

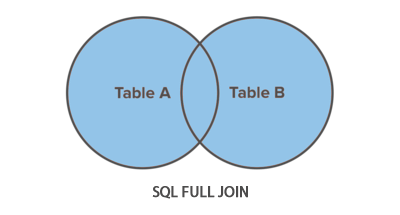
**SQL FULL JOIN Statement**

In this tutorial you will learn how to retrieve data from two tables using SQL full join.

**Using Full Joins**

A FULL JOIN returns all the rows from the joined tables, whether they are matched or not i.e. you can say a full join combines the functions of a [LEFT JOIN](https://www.tutorialrepublic.com/sql-tutorial/sql-left-join-operation.php) and a [RIGHT JOIN](https://www.tutorialrepublic.com/sql-tutorial/sql-right-join-operation.php). Full join is a type of [outer join](https://www.tutorialrepublic.com/sql-tutorial/sql-joining-tables.php#outer-join) that's why it is also referred as *full outer join*.

The following Venn diagram illustrates how full join works.



**Note:** An outer join is a join that includes rows in a result set even though there may not be a match between rows in the two tables being joined.

To understand this clearly, let's look at the following *employees* and *departments* tables.

|  |  |  |
| --- | --- | --- |
| +--------+--------------+------------+---------+  | emp\_id | emp\_name | hire\_date | dept\_id |  +--------+--------------+------------+---------+  | 1 | Ethan Hunt | 2001-05-01 | 4 |  | 2 | Tony Montana | 2002-07-15 | 1 |  | 3 | Sarah Connor | 2005-10-18 | 5 |  | 4 | Rick Deckard | 2007-01-03 | 3 |  | 5 | Martin Blank | 2008-06-24 | NULL |  +--------+--------------+------------+---------+ |  | +---------+------------------+  | dept\_id | dept\_name |  +---------+------------------+  | 1 | Administration |  | 2 | Customer Service |  | 3 | Finance |  | 4 | Human Resources |  | 5 | Sales |  +---------+------------------+ |
| Table: **employees** |  | Table: **departments** |

Now, let's say you just want to retrieve the names of all the employees and the names of available departments, regardless of whether they have corresponding rows in the other table, in that case you can use a full join as demonstrated below.

The following statement retrieves all the departments as well as the details of all the employees by joining the *employees* and *departments* tables together using the common *dept\_id* field.

**Example**

[**Try this code »**](javascript:void(0);)

SELECT t1.emp\_id, t1.emp\_name, t1.hire\_date, t2.dept\_name

FROM employees AS t1 FULL JOIN departments AS t2

ON t1.dept\_id = t2.dept\_id ORDER BY emp\_name;

Some databases, such as Oracle, MySQL do not support full joins. In that case you can use the UNION ALL operator to combine the LEFT JOIN and RIGHT JOIN as follows:

**Example**

[**Try this code »**](javascript:void(0);)

SELECT t1.emp\_id, t1.emp\_name, t1.hire\_date, t2.dept\_name

FROM employees AS t1 LEFT JOIN departments AS t2

ON t1.dept\_id = t2.dept\_id

UNION ALL

SELECT t1.emp\_id, t1.emp\_name, t1.hire\_date, t2.dept\_name

FROM employees AS t1 RIGHT JOIN departments AS t2

ON t1.dept\_id = t2.dept\_id ORDER BY emp\_name;

After executing the above command, you'll get the output something like this:

+--------+--------------+------------+------------------+

| emp\_id | emp\_name | hire\_date | dept\_name |

+--------+--------------+------------+------------------+

| NULL | NULL | NULL | Customer Service |

| 1 | Ethan Hunt | 2001-05-01 | Human Resources |

| 1 | Ethan Hunt | 2001-05-01 | Human Resources |

| 5 | Martin Blank | 2008-06-24 | NULL |

| 4 | Rick Deckard | 2007-01-03 | Finance |

| 4 | Rick Deckard | 2007-01-03 | Finance |

| 3 | Sarah Connor | 2005-10-18 | Sales |

| 3 | Sarah Connor | 2005-10-18 | Sales |

| 2 | Tony Montana | 2002-07-15 | Administration |

| 2 | Tony Montana | 2002-07-15 | Administration |

+--------+--------------+------------+------------------+

As you can see the result includes all the rows from both the *departments* and *employees* table.

**Tip:** In a join query, the left table is the one that appears leftmost in the JOIN clause, and the right table is the one that appears rightmost.

**Note:** When performing outer joins, wherever the DBMS (Database Management System) can't match any row, it places NULL in the columns to indicate data do not exist.

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-right-join-operation.php)

**SQL UNION Operation**

In this tutorial you will learn how to combine the results of two or more SQL queries.

**The UNION Operator**

The UNION operator is used to combine the results of two or more SELECT queries into a single result set. The union operation is different from using joins that combine columns from two tables. The union operation creates a new table by placing all rows from two source tables into a single result table, placing the rows on top of one another.

These are basic rules for combining the result sets of two SELECT queries by using UNION:

* The number and the order of the columns must be the same in all queries.
* The data types of the corresponding columns must be compatible.

When these criteria are met, the tables are *union-compatible*:

**Syntax**

The basic syntax of UNION can be given with:

SELECT *column\_list* FROM *table1\_name*  
UNION SELECT *column\_list* FROM *table2\_name*;

To understand the union operation in a better way, let's assume that some hypothetical fields, like *first\_name* and *last\_name* exists in our *employees* and *customers* tables. Please note that these fields do not actually exist in our *demo* database tables.

|  |  |  |
| --- | --- | --- |
| +----+------------+-----------+--------+  | id | first\_name | last\_name | salary |  +----+------------+-----------+--------+  | 1 | Ethan | Hunt | 5000 |  | 2 | Tony | Montana | 6500 |  | 3 | Sarah | Connor | 8000 |  | 4 | Rick | Deckard | 7200 |  | 5 | Martin | Blank | 5600 |  +----+------------+-----------+--------+ |  | +----+------------+-----------+----------+  | id | first\_name | last\_name | city |  +----+------------+-----------+----------+  | 1 | Maria | Anders | Berlin |  | 2 | Fran | Wilson | Madrid |  | 3 | Dominique | Perrier | Paris |  | 4 | Martin | Blank | Turin |  | 5 | Thomas | Hardy | Portland |  +----+------------+-----------+----------+ |
| Table: **employees** |  | Table: **customers** |

Let's perform a union operation to combine the results of two queries.

The following statement returns the first and last names of all the customers and employees:

**Example**

[**Try this code »**](javascript:void(0);)

SELECT first\_name, last\_name FROM employees

UNION

SELECT first\_name, last\_name FROM customers;

After executing the above statement, the result set will look something like this:

+---------------+--------------+

| first\_name | last\_name |

+---------------+--------------+

| Ethan | Hunt |

| Tony | Montana |

| Sarah | Connor |

| Rick | Deckard |

| Martin | Blank |

| Maria | Anders |

| Fran | Wilson |

| Dominique | Perrier |

| Thomas | Hardy |

+---------------+--------------+

The UNION operation eliminates the duplicate rows from the combined result set, by default. That's why the above query returns only 9 rows, because if you notice the name "Martin Blank" appears in both the *employees* and *customers* tables.

However, if you want to keep the duplicate rows you can use the ALL keyword, as follow:

**Example**

[**Try this code »**](javascript:void(0);)

SELECT first\_name, last\_name FROM employees

UNION ALL

SELECT first\_name, last\_name FROM customers;

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-cross-join-operation.php)

**SQL LIKE Operator**

In this tutorial you will learn how to retrieve the data based on a partial match.

**Pattern Matching**

So far, you've seen the conditions that identify an exact string, e.g. WHERE name='Lois Lane'. But in SQL you can perform partial or pattern matching too using the LIKE operator.

The LIKE operator provides a measure of pattern matching by allowing you to specify wildcards for one or more characters. You can use the following two wildcard characters:

* The percent sign (%) — Matches any number of characters, even zero characters.
* The underscore (\_) — Matches exactly one character

Here're some examples that show how to use the LIKE operator with wildcards.

| **Statement** | **Meaning** | **Values Returned** |
| --- | --- | --- |
| WHERE name LIKE 'Da%' | Find names beginning with *Da* | David, Davidson |
| WHERE name LIKE '%th' | Find names ending with *th* | Elizabeth, Smith |
| WHERE name LIKE '%on%' | Find names containing the *on* | Davidson, Toni |
| WHERE name LIKE 'Sa\_' | Find names beginning with *Sa* and is followed by at most one character | Sam |
| WHERE name LIKE '\_oy' | Find names ending with *oy* and is preceded by at most one character | Joy, Roy |
| WHERE name LIKE '\_an\_' | Find names containing *an* and begins and ends with at most one character | Dana, Hans |
| WHERE name LIKE '%ar\_' | Find names containing *ar*, begins with any number of characters, and ends with at most one character | Richard, Karl |
| WHERE name LIKE '\_ar%' | Find names containing *ar*, begins with at most one character, and ends with any number of characters | Karl, Mariya |

Let's put the statements we've discussed above into real use by searching some records.

Consider we've an *employees* table in our database with the following records:

+--------+------------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+------------------+------------+--------+---------+

| 1 | Ethan Hunt | 2001-05-01 | 5000 | 4 |

| 2 | Tony Montana | 2002-07-15 | 6500 | 1 |

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 4 | Rick Deckard | 2007-01-03 | 7200 | 3 |

| 5 | Martin Blank | 2008-06-24 | 5600 | NULL |

| 6 | simons bistro | 2009-04-01 | 6000 | 1 |

+--------+------------------+------------+--------+---------+

Now, let's say you want to find out all the employees whose name begins with S letter.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=pattern-matching-with-like-operator)

SELECT \* FROM employees

WHERE emp\_name LIKE 'S%';

After executing the query, you'll get the output something like this:

+--------+------------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+------------------+------------+--------+---------+

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

| 6 | simons bistro | 2009-04-01 | 6000 | 1 |

+--------+------------------+------------+--------+---------+

In MySQL nonbinary string (CHAR, VARCHAR, TEXT) comparisons are case-insensitive by default, whereas binary strings (BINARY, VARBINARY, BLOB) comparisons are case-sensitive.

This means that if you search with WHERE name LIKE 'S%', you get all column values that start with S or s (as you can see we've got both "Sarah" and "simons"). However, if you want to make this search case sensitive you can use the BINARY operator as follow:

**Example**

[**Try this code »**](javascript:void(0);)

-- Syntax for MySQL Database

SELECT \* FROM employees

WHERE BINARY emp\_name LIKE 'S%';

Now, this statement will return only those employees whose name starts with capital S letter:

+--------+------------------+------------+--------+---------+

| emp\_id | emp\_name | hire\_date | salary | dept\_id |

+--------+------------------+------------+--------+---------+

| 3 | Sarah Connor | 2005-10-18 | 8000 | 5 |

+--------+------------------+------------+--------+---------+

**Note:** If you want a column always to be treated in case-sensitive fashion, declare it with a case sensitive or binary collation to avoid any performance issue.

**Tip:** Partial matches are useful when you don't know the exact form of the string for which you're searching. You can also use partial matching to retrieve multiple rows that contain similar strings in one of the table's columns.

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-union-operation.php)[NEXT PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-alter-table-statement.php)

ADVERTISEMENTS

**SQL ALTER TABLE Statement**

In this tutorial you will learn how to alter or modify an existing table using SQL.

**Modifying Existing Tables**

It is quite possible that after creating a table, as you start using it, you may discover you've forgot to mention any column or constraint or specified a wrong name for the column.

In such situation you can use the ALTER TABLE statement to alter or change an existing table by adding, changing, or deleting a column in the table.

Consider we've a *shippers* table in our database, whose structure is as follows:

+--------------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+-------------+------+-----+---------+----------------+

| shipper\_id | int | NO | PRI | NULL | auto\_increment |

| shipper\_name | varchar(60) | NO | | NULL | |

| phone | varchar(60) | NO | | NULL | |

+--------------+-------------+------+-----+---------+----------------+

We'll use this *shippers* table for all of our ALTER TABLE statements.

Now suppose that we want to expand the existing *shippers* table by adding one more column. But, the question is how we can do this using SQL commands? Well let's find out.

**Adding a New Column**

The basic syntax for adding a new column to an existing table can be given with:

ALTER TABLE *table\_name* ADD *column\_name* *data\_type* *constraints*;

The following statement adds a new column *fax* to the *shippers* table.

**Example**

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=sql&file=add-a-new-column-to-the-table)

ALTER TABLE shippers ADD fax VARCHAR(20);

Now, after executing the above statement if you see the table structure using the command DESCRIBE shippers; on MySQL command-line, it looks as follow:

+--------------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+-------------+------+-----+---------+----------------+

| shipper\_id | int | NO | PRI | NULL | auto\_increment |

| shipper\_name | varchar(60) | NO | | NULL | |

| phone | varchar(60) | NO | | NULL | |

| fax | varchar(20) | YES | | NULL | |

+--------------+-------------+------+-----+---------+----------------+

**Note:** If you want to add a NOT NULL column to an existing table then you must specify an explicit [default value](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php#default). This default value is used to populate the new column for every row that already exists in your table.

**Tip:** When adding a new column to the table, if neither NULL nor NOT NULL is specified, the column is treated as though NULL had been specified.

MySQL add new columns at the end by default. However, if you want to add a new column after a specific column you can use the AFTER clause, as follow:

mysql> ALTER TABLE shippers ADD fax VARCHAR(20) AFTER shipper\_name;

MySQL provide another clause FIRST that you can use to add a new column at first place within a table. Just replace the clause AFTER with FIRST in the previous example to add the column *fax* at the beginning of the *shippers* table.

**Changing Column Position**

In MySQL, if you've already created a table but unhappy with the existing column position within the table, you can change it any time using the following syntax:

ALTER TABLE *table\_name*  
MODIFY *column\_name* *column\_definition* AFTER *column\_name*;

The following statement place the column *fax* after *shipper\_name* column in *shippers* table.

mysql> ALTER TABLE shippers MODIFY fax VARCHAR(20) AFTER shipper\_name;

**Adding Constraints**

Our current shippers table has one major problem. If you insert records with duplicate phone numbers it wouldn't stop you from doing that, which is not good, it should be unique.

You can fix this by adding a constraint UNIQUE to the *phone* column. The basic syntax for adding this constraint to existing table columns can be given with:

ALTER TABLE *table\_name* ADD UNIQUE (*column\_name*,...);

The following statement adds a constraint UNIQUE to the *phone* column.

mysql> ALTER TABLE shippers ADD UNIQUE (phone);

After executing this statement if you try to insert a duplicate phone number, you'll get an error.

Similarly, if you've created a table without a PRIMARY KEY, you can add one with:

ALTER TABLE *table\_name* ADD PRIMARY KEY (*column\_name*,...);

The following statement adds a constraint [PRIMARY KEY](https://www.tutorialrepublic.com/sql-tutorial/sql-constraints.php#primary-key) to the *shipper\_id* column, if not defined.

mysql> ALTER TABLE shippers ADD PRIMARY KEY (shipper\_id);

**Removing Columns**

The basic syntax for removing a column from an existing table can be given with:

ALTER TABLE *table\_name* DROP COLUMN *column\_name*;

The following statement removes our newly added column *fax* from the *shippers* table.

mysql> ALTER TABLE shippers DROP COLUMN fax;

Now, after executing the above statement if you see the table structure, it looks as follow:

+--------------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+--------------+-------------+------+-----+---------+----------------+

| shipper\_id | int | NO | PRI | NULL | auto\_increment |

| shipper\_name | varchar(60) | NO | | NULL | |

| phone | varchar(20) | NO | UNI | NULL | |

+--------------+-------------+------+-----+---------+----------------+

**Changing Data Type of a Column**

You can modify the data type of a column in SQL Server by using the ALTER clause, as follow:

ALTER TABLE *table\_name* ALTER COLUMN *column\_name* *new\_data\_type*;

The MySQL database server however does not support the ALTER COLUMN syntax. It supports an alternate MODIFY clause that you can use to modify the column as follows:

ALTER TABLE *table\_name* MODIFY *column\_name* *new\_data\_type*;

The following statement changes the current data type of the *phone* column in our *shippers* table from VARCHAR to CHAR and length from 20 to 15.

mysql> ALTER TABLE shippers MODIFY phone CHAR(15);

Similarly, you can use the MODIFY clause to switch between whether a column in the MySQL table should allow null values or not by re-specifying the existing column definition and add the NULL or NOT NULL constraint at the end, like this:

mysql> ALTER TABLE shippers MODIFY shipper\_name CHAR(15) NOT NULL;

**Renaming Tables**

The basic syntax for renaming an existing table in MySQL can be given with:

ALTER TABLE *current\_table\_name* RENAME *new\_column\_name*;

The following statement renames our *shippers* table *shipper*.

mysql> ALTER TABLE shippers RENAME shipper;

You can also achieve the same thing in MySQL using the RENAME TABLE statement, as follow:

mysql> RENAME TABLE shippers TO shipper;

[PREVIOUS PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-like-operator.php)[NEXT PAGE](https://www.tutorialrepublic.com/sql-tutorial/sql-aliases.php)

ADVERTISEMENTS