# **MySQL CREATE INDEX**

**Summary**: in this tutorial, you will learn about the index and how to use the MySQL CREATE INDEX statement to add an index to a table.

## **The phone book analogy**

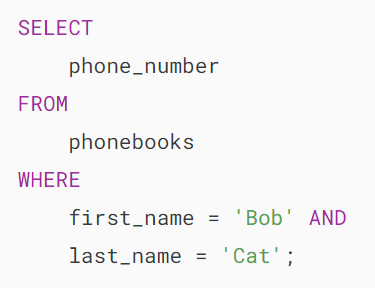
Suppose you have a phone book that contains all the names and phone numbers of people in a city.

Let’s say you want to find Bob Cat’s phone number. Knowing that the names are alphabetically ordered, you first look for the page where the last name is Cat, then you look for Bob and his phone number.

If the names in the phone book were not sorted alphabetically, you would need to go through all the pages, reading every name on it until you find Bob Cat.

This is called **sequential searching**. You go over all the entries until you find the person with the phone number that you are looking for.

Relating the phone book to the table, if you have the table phonebooks and you need to find the phone number of Bob Cat, you would perform the following query:



It is pretty easy. Although the query is fast, the database has to scan all the rows of the table until it finds the row. If the table has millions of rows, without an index, the data retrieval would take a lot of time to return the result.

## **Introduction to Database Indexes**

An index is a data structure such as a B-Tree that improves the speed of data retrieval on a table at the cost of additional writes and storage to maintain it.

The query optimizer may use indexes to quickly locate data without having to scan every row in a table for a given query.

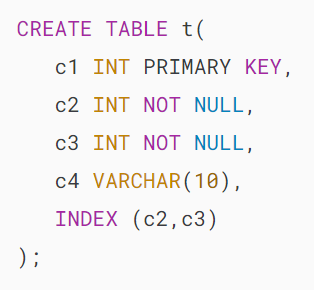
When you [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) with a [primary key](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/) or[unique key](https://www.mysqltutorial.org/mysql-basics/mysql-unique-constraint/), MySQL automatically creates a special index named PRIMARY. This index is called the [clustered index](https://www.mysqltutorial.org/mysql-index/mysql-clustered-index/).

The PRIMARY index is special because the index itself is stored together with the data in the same table. The clustered index enforces the order of rows in the table.

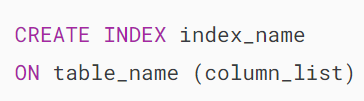
Other indexes other than the PRIMARY index are called secondary indexes or non-clustered indexes.

## **MySQL CREATE INDEX statement**

Typically, you create indexes for a table at the time of creation. For example, the following statement [creates a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) with an index that consists of two columns c2 and c3.



To add an index for a column or a set of columns, you use the CREATE INDEX statement as follows:



To create an index for a column or a list of columns, you specify the index name, the table to which the index belongs, and the column list.

For example, to add a new index for the column c4, you use the following statement:



By default, MySQL creates the B-Tree index if you don’t specify the index type.

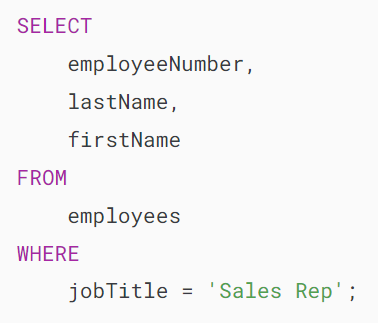
The following table shows the valid index types based on the [storage engine](https://www.mysqltutorial.org/mysql-administration/mysql-storage-engines/) of the table:

| **Storage Engine** | **Allowed Index Types** |
| --- | --- |
| InnoDB | BTREE |
| MyISAM | BTREE |
| MEMORY/HEAP | HASH, BTREE |

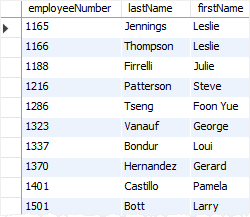
Notice that the CREATE INDEX statement above is a simplified version of the CREATE INDEX statement introduced by MySQL. We will cover more options in the subsequent tutorials.

## **MySQL CREATE INDEX example**

The following statement finds employees whose job title is Sales Rep:



Here is the output:



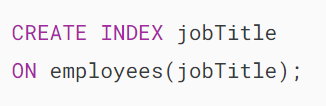
We have 17 rows indicating that 17 employees whose job title is the Sales Rep.

To see how MySQL internally performed this query, you add the EXPLAIN clause at the beginning of the SELECT statement as follows:

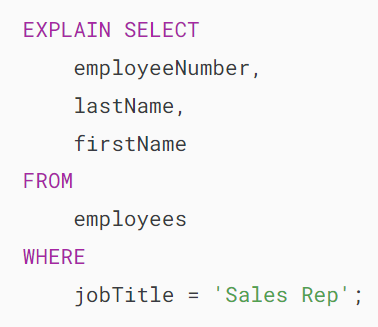
MySQL Index EXPLAIN statement

As you can see, MySQL had to scan the whole table which consists of 23 rows to find the employees with the Sales Rep job title.

Now, let’s create an index for the  jobTitle column by using the CREATE INDEX statement:



Execute the EXPLAIN statement again:

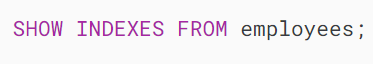


The output is:

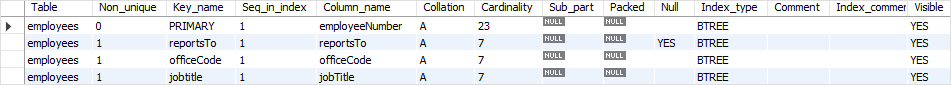


The output shows that MySQL just had to locate 17 rows from the  jobTitle index as indicated in the key column without scanning the whole table.

To list all indexes of a table, you use the [SHOW INDEXES](https://www.mysqltutorial.org/mysql-index/mysql-show-indexes/) statement, for example:



Here is the output:



## **Summary**

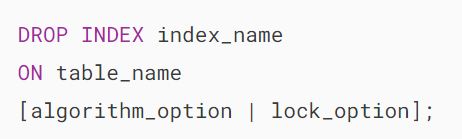
* A database index enhances retrieval speed but comes with the cost of increased writing overhead.
* Use the CREATE INDEX statement to create a new index for a table.

# **MySQL DROP INDEX**

**Summary**: in this tutorial, you will learn how to use the MySQL DROP INDEX statement to remove existing indexes of a table.

## **Introduction to MySQL DROP INDEX statement**

To delete an existing index from a table, you use the DROP INDEX statement as follows:



In this syntax:

* First, specify the name of the index which you want to remove after the DROP INDEX keywords.
* Second, specify the name of the table to which the index belongs.

### **Algorithm**

The algorithm\_option allows you to specify a specific algorithm used for the index removal.

The following shows the syntax of the algorithm\_option clause:



For the index removal, the following algorithms are supported:

* COPY: MySQL copies data from the original table to a new table row by row. After completing the copy, the DROP INDEX executes on the newly created table. Please note that during the copy process, you’re not allowed to perform any concurrent data manipulation statements such as [INSERT](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) and [UPDATE](https://www.mysqltutorial.org/mysql-basics/mysql-update/).
* INPLACE: The INPLACE algorithm involves rebuilding the table directly in its existing location without creating a new copy of the table. During both the preparation and execution phases of the index removal process, MySQL places an exclusive metadata lock on the table. This allows concurrent data manipulation statements to be executed alongside the index removal process.

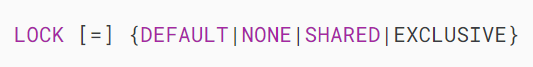
Note that the ALGORITHM clause is optional. If you skip it, MySQL uses INPLACE. In case the INPLACE is not supported, MySQL uses COPY.

Using DEFAULT has the same effect as omitting the ALGORITHM clause.

### **Lock**

The lock\_option controls the level of concurrent reads and writes on the table while the index is being removed.

The following shows the syntax of the lock\_option:

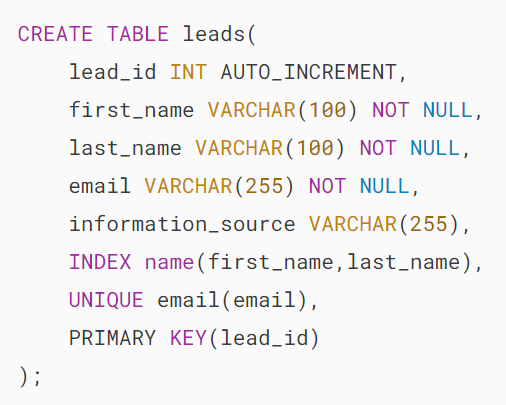


The following locking modes are supported:

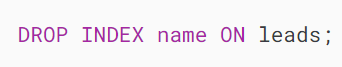
* DEFAULT: This allows you to have the maximum level of concurrency for a given algorithm. First, it allows concurrent reads and writes if supported. If not, allow concurrent reads if supported. If not, enforce exclusive access.
* NONE: If the NONE is supported, you can have concurrent reads and writes. Otherwise, MySQL issues an error.
* SHARED: If the SHARED is supported, you can have concurrent reads but not writes. MySQL issues an error if the concurrent reads are not supported.
* EXCLUSIVE: This enforces exclusive access.

## **MySQL DROP INDEX statement examples**

Let’s [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) for the demonstration:



The following statement removes the name index from the leads table:



The following statement drops the email index from the leads table with a specific algorithm and lock:

## 

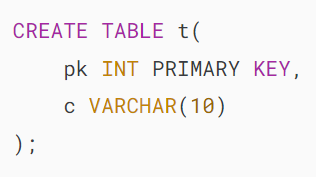
## **MySQL DROP PRIMARY KEY index**

To drop the [primary key](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/) whose index name is PRIMARY, you use the following statement:



Notice that the PRIMARY is a reversed word in MySQL. However, the index name is PRIMARY. Therefore, you need to place the PRIMARY inside the quotes to specify the primary index.

The following statement creates a new table named t with a primary key:



To drop the primary key index, you use the following statement:

## 

## **Summary**

* Use the DROP INDEX statement to remove an existing index.
* Use the DROP INDEX `PRIMARY` to remove the primary index.

# **MySQL SHOW INDEXES**

**Summary**: in this tutorial, you will learn how to query index information from a table by using the MySQL SHOW INDEXES command.

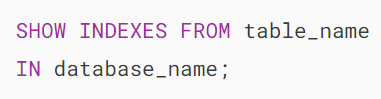
## **Introduction to MySQL SHOW INDEXES command**

To query the index information of a table, you use the SHOW INDEXES statement as follows:

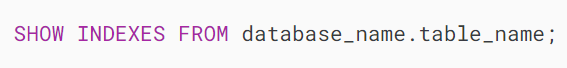


To get the index of a table, you specify the table name after the FROM keyword. The statement will return the index information associated with the table in the current database.

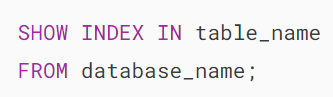
You can specify the database name if you are not connected to any database or you want to get the index information of a table in a different database:



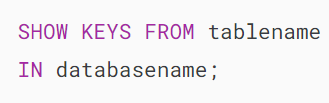
The following query is similar to the one above:



Note that INDEX and KEYS are the synonyms of the INDEXES, IN is the synonym of the FROM, therefore, you can use these synonyms in the SHOW INDEXES column instead. For example:



Or



The SHOW INDEXES returns the following information:

### **table**

The name of the table

### **non\_unique**

1 if the index can contain duplicates, 0 if it cannot.

### **key\_name**

The name of the index. The [primary key](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/) index always has the name of PRIMARY.

### **seq\_in\_index**

The column sequence number in the index. The first column sequence number starts from 1.

### **column\_name**

The column name

### **collation**

Collation represents how the column is sorted in the index. A means ascending, B means descending, or NULL means not sorted.

### **cardinality**

The cardinality returns an estimated number of unique values in the index.

Note that the higher the cardinality, the greater the chance that the query optimizer uses the index for lookups.

### **sub\_part**

The index prefix. It is null if the entire column is indexed. Otherwise, it shows the number of indexed characters in case the column is partially indexed.

### **packed**

indicates how the key is packed; NUL if it is not.

### **null**

YES if the column contains NULL values and blank if it does not.

### **index\_type**

represents the index method used such as BTREE, HASH, RTREE, or [FULLTEXT](https://www.mysqltutorial.org/mysql-full-text-search/mysql-full-text-index/).

### **comment**

The information about the index is not described in its own column.

### **index\_comment**

shows the comment for the index specified when you create the index with the COMMENT attribute.

### **visible**

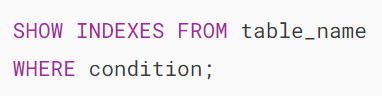
Whether the index is visible or invisible to the query optimizer or not; YES if it is, NO if not.

### **expression**

If the index uses an expression rather than a column or column prefix value, the expression indicates the expression for the key part and also the column\_name column is NULL.

## **Filter index information**

To filter index information, you use a [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/) clause as follows:

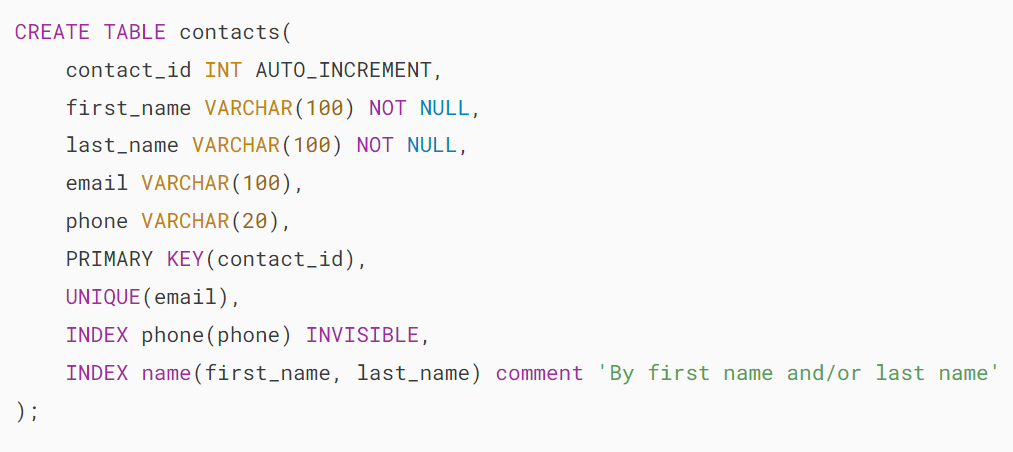


You can use any information returned by the SHOW INDEXES statement to filter the index information. For example, the following statement returns only the invisible indexes of a table:

### 

### **MySQL SHOW INDEXES examples**

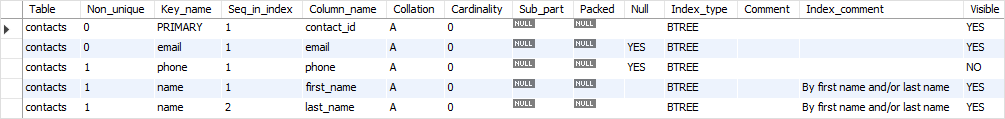
We will [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named contacts to demonstrate the SHOW INDEXES command:



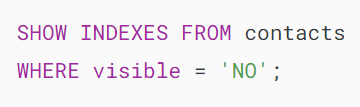
The following command returns all index information from the contacts table:



The output is:



To get the invisible indexes of the contacts table, you add a [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/) clause as follows:



Here is the output:

MySQL SHOW INDEXES - get invisible indexes

# **MySQL Prefix Index**

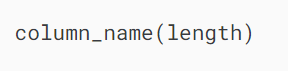
**Summary**: in this tutorial, you will learn how to use MySQL prefix index to create indexes for character string columns.

## **Introduction to MySQL Prefix Index**

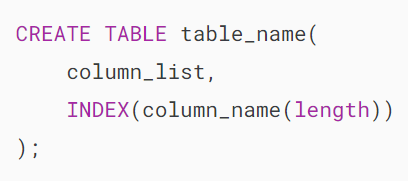
When you [create a secondary index](https://www.mysqltutorial.org/mysql-index/mysql-create-index/) for a column, MySQL stores the values of the columns in a separate data structure e.g., B-Tree and Hash.

In case the columns are string columns, the index will consume a lot of disk space and potentially slow down the [INSERT](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) operations.

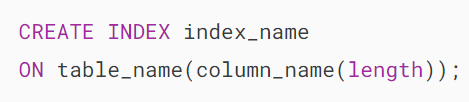
To address this issue, MySQL allows you to create an index for the leading part of the column values of the string columns using the following syntax:



For example, the following statement creates the column prefix key part at the time of table creation:



Or add an index to an existing table:



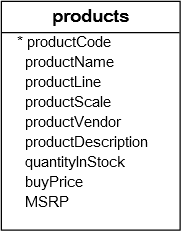
In this syntax, the length is the number of characters for the non-binary string types such as [CHAR](https://www.mysqltutorial.org/mysql-basics/mysql-char-data-type/), [VARCHAR](https://www.mysqltutorial.org/mysql-basics/mysql-varchar/), and [TEXT](https://www.mysqltutorial.org/mysql-basics/mysql-text/) and the number of bytes for binary string types e.g., BINARY, VARBINARY, and BLOB.

MySQL allows you to optionally create column prefix key parts for CHAR, VARCHAR, BINARY, and VARBINARY columns. If you create indexes for BLOB and TEXT columns, you must specify the column prefix key parts.

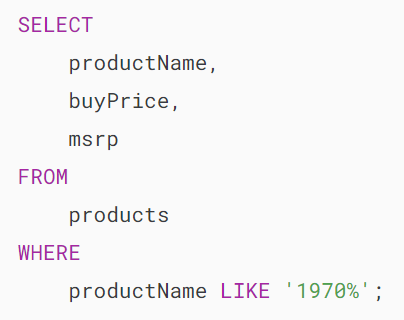
Notice that the prefix support and lengths of prefixes if supported are [storage engine](https://www.mysqltutorial.org/mysql-administration/mysql-storage-engines/) dependent. For InnoDB tables with  REDUNDANT or COMPACT row format, the maximum prefix length is 767 bytes. However, for the InnoDB tables with  DYNAMIC or COMPRESSED row format, the prefix length is 3,072 bytes. MyISAM tables have a prefix length of up to 1,000 bytes.

## **MySQL prefix index example**

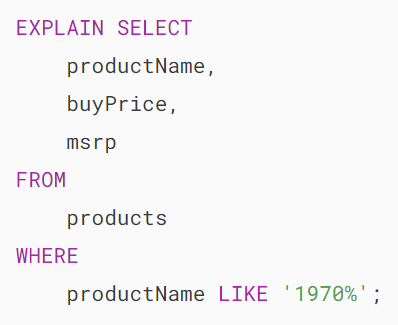
We will use the products table from the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/) for the demonstration.



The following query finds the products whose names start with the string 1970:



Because there is no index for the  productName column, the query optimizer has to scan all rows to return the result as shown in the output of the EXPLAIN statement below:



Here is the output:

MySQL Prefix Index Example

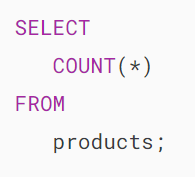
If you often find the products by the product name, then you should [create an index](https://www.mysqltutorial.org/mysql-index/mysql-create-index/) for this column because it will be more efficient for searches.

The size of the product name column is 70 characters. We can use the column prefix key parts.

The next question is how do you choose the length of the prefix? To do this, you can investigate the existing data. The goal is to maximize the uniqueness of the values in the column when you use the prefix.

To do this, you follow these steps:

Step 1. Find the number of rows in the table:



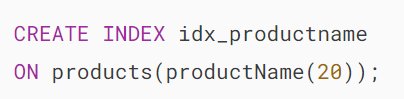
Step2. Evaluate different prefix lengths until you can achieve the reasonable uniqueness of rows:



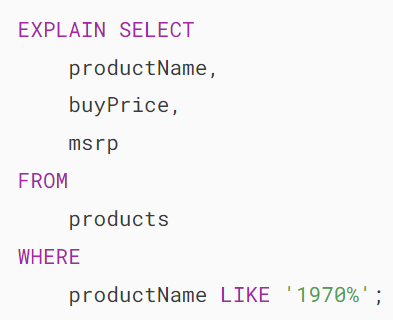
MySQL Prefix Index - row count

As shown in the output, 20 is a good prefix length in this case because if we use the first 20 characters of the product name for the index, all product names are unique.

Let’s create an index with the prefix length 20 for the productName column:



And execute the query that finds products whose name starts with the string 1970 again:



MySQL Prefix Index - result

Now, the query optimizer uses the newly created index which is much faster and more efficient than before.

# **MySQL UNIQUE Index**

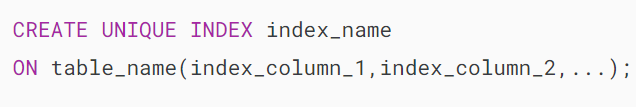
**Summary**: in this tutorial, you will learn how to use the MySQL UNIQUE index to prevent duplicate values in one or more columns in a table.

## **Introduction to the MySQL UNIQUE index**

To enforce the uniqueness value of one or more columns, you often use the [PRIMARY KEY](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/) constraint. However, each table can have only one primary key. So if you want to have more than one column or a set of columns with unique values, you cannot use the primary key constraint.

Luckily, MySQL provides another kind of [index](https://www.mysqltutorial.org/mysql-index/mysql-create-index/) called UNIQUE index that allows you to enforce the uniqueness of values in one or more columns. Unlike the PRIMARY KEY index, you can have more than one UNIQUE index per table.

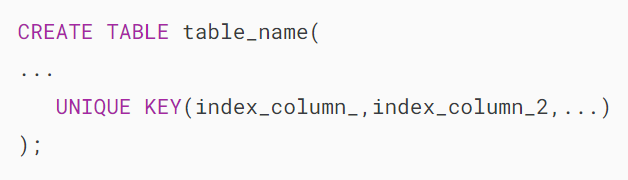
To create a UNIQUE index, you use the CREATE UNIQUE INDEX statement as follows:



Another way to enforce the uniqueness of value in one or more columns is to use the [UNIQUE](https://www.mysqltutorial.org/mysql-basics/mysql-unique-constraint/) constraint.

When you create a [UNIQUE constraint](https://www.mysqltutorial.org/mysql-basics/mysql-unique-constraint/), MySQL creates a UNIQUE index behind the scenes.

The following statement illustrates how to create a unique constraint when you [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/).



In this statement, you can also use the UNIQUE INDEX instead of the UNIQUE KEY because they are synonyms.

If you want to add a unique constraint to an existing table, you can use the ALTER TABLE statement as follows:

## 

## **MySQL UNIQUE Index & NULL**

Unlike other database systems, MySQL considers [NULL](https://www.mysqltutorial.org/mysql-basics/mysql-null/) values as distinct values. Therefore, you can have multiple NULL values in the UNIQUE index.

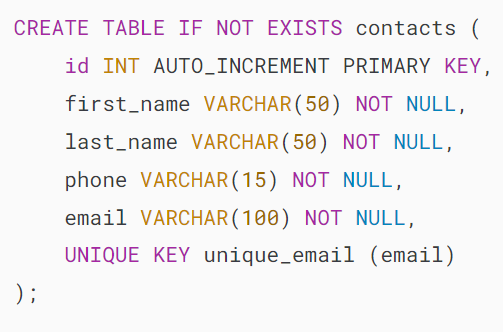
This is how MySQL was designed. It is not a bug even though it was reported as a [bug](http://bugs.mysql.com/bug.php?id=25544).

Another important point is that the UNIQUE constraint does not apply to NULL values except for the BDB [storage engine](https://www.mysqltutorial.org/mysql-administration/mysql-storage-engines/).

## **MySQL UNIQUE index examples**

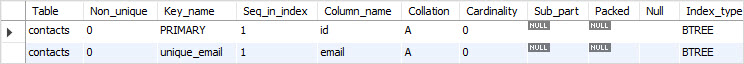
Suppose you want to manage contacts in an application. You also want the email of every contact in the contacts table must be unique.

To enforce this rule, you create a unique constraint in the [CREATE TABLE](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) statement as follows:

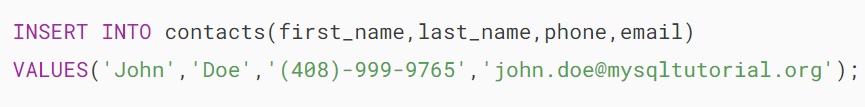


If you use the [SHOW INDEXES](https://www.mysqltutorial.org/mysql-index/mysql-show-indexes/) statement, you will see that MySQL created a UNIQUE index for email column.

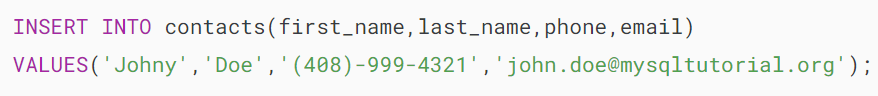




Let’s [insert a row](https://www.mysqltutorial.org/mysql-basics/mysql-insert/) into the contacts table.

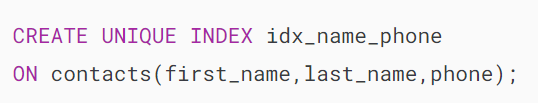


Now if you try to insert a row whose email is john.doe@mysqltutorial.org, you will get an error message.

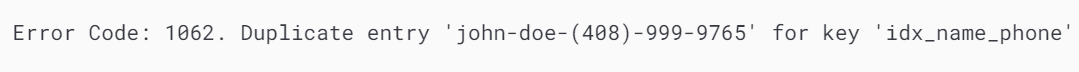
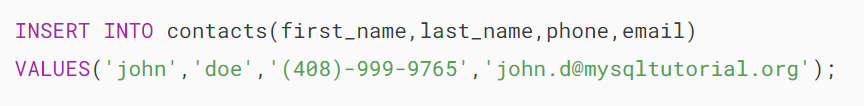




Suppose you want a combination of first\_name, last\_name, and  phone is also unique among contacts. In this case, you use the CREATE INDEX statement to create a UNIQUE index for those columns as follows:



Adding the following row to the contacts table causes an error because the combination of the first\_name, last\_name, and phone already exists.



# **MySQL Composite Index**

**Summary**: in this example, you will learn about the MySQL composite index and how to use it to speed up your queries.

## **Introduction to MySQL Composite Index**

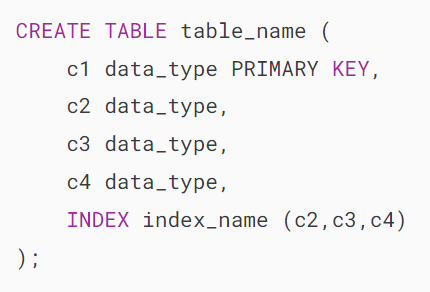
A composite index is an index on multiple columns. MySQL allows you to create a composite index that consists of up to 16 columns.

A composite index is also known as a multiple-column index.

The query optimizer uses the composite indexes for queries that test all columns in the index, or queries that test the first columns, the first two columns, and so on.

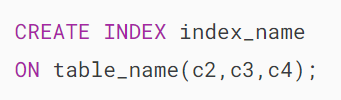
If you specify the columns in the correct order in the index definition, a single composite index can enhance the performance of queries involving those columns on the same table.

To create a composite index during table creation, you use the following statement:

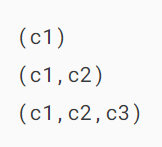


In this syntax, the composite index consists of three columns c2, c3, and c4.

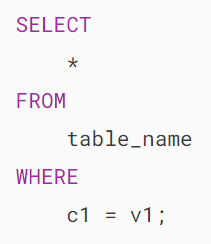
Alternatively, you can add a composite index to an existing table using the CREATE INDEX statement:

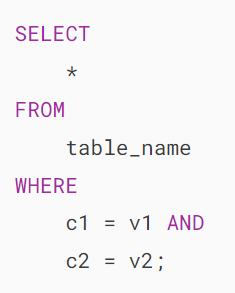


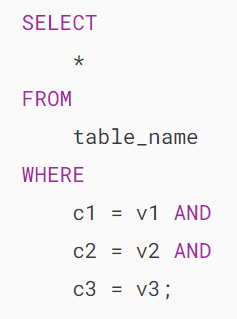
Notice that if you have a composite index on (c1,c2,c3), you will have indexed search capabilities on one of the following column combinations:



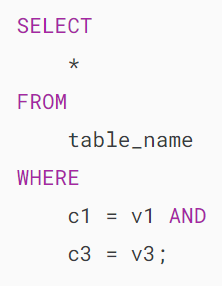
For example:





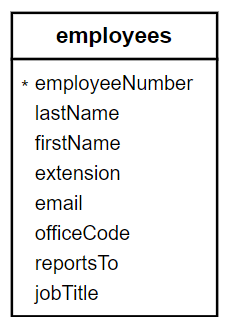


The query optimizer cannot use the index to perform lookups if the columns do not form the leftmost prefix of the index. For example, the following queries cannot use the composite for lookups:

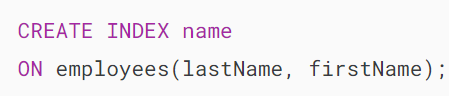


## **MySQL composite index example**

We will use the employees table in the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/) for the demonstration:



The following statement creates a composite index over the lastName and firstName columns:



First, the name index can be used for lookups in the queries that specify a lastName value because the lastName column is the leftmost prefix of the index.

Second, the name index can be used for queries that specify values for the combination of the lastName and firstName values.

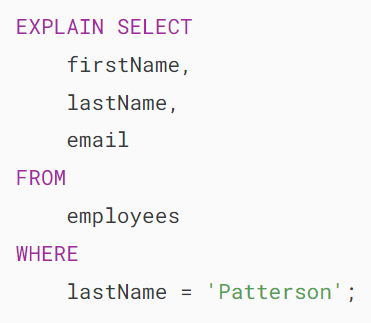
The name index, therefore, is used for lookups in the following queries:

1) Find employees whose last name is Patterson



This query uses the name index because the leftmost prefix of the index, which is the lastName column, is used for lookups.

You can verify this by adding the EXPLAIN clause to the query:



Here is the output:

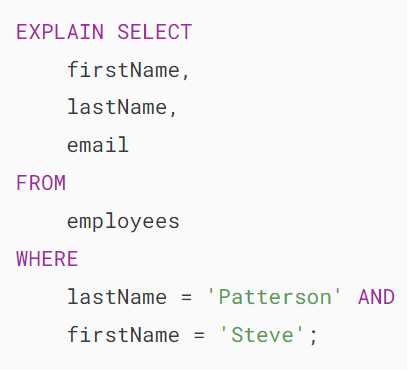
MySQL Composite Index Example 1

2) Find employees whose last name is Patterson and the first name is Steve:



In this query, both lastName and firstName columns are used for lookups, therefore, it uses the name index.

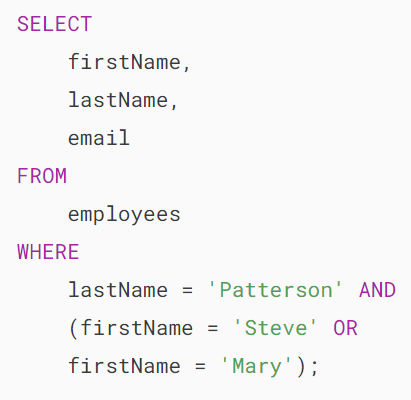
Let’s verify it:



The output is:

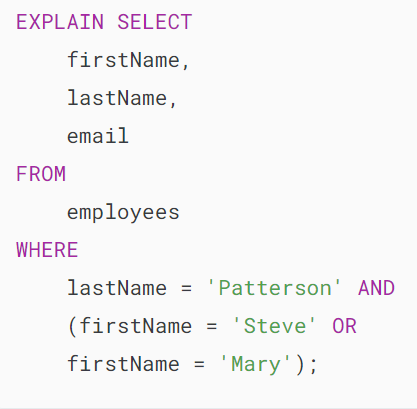
MySQL Composite Index Example 2

3) Find employees whose last name is Patterson and the first name is Steve or Mary:



This query is similar to the second one which both lastName and firstName columns are used for lookups.

The following statement verifies the index usage:



The output is:

MySQL Composite Index Example 3

The query optimizer cannot use the name index for lookups in the following queries because only the firstName column which is not the leftmost prefix of the index is used:



Similarly, the query optimizer cannot use the name index for the lookups in the following query because either the firstName or lastName column is used for lookups.



## **Summary**

* Composite indexes are indexes that involve more than one column.
* Define composite indexes when your queries involve conditions or sorting on multiple columns.
* Using composite indexes properly can significantly improve the performance of queries that filter or sort based on the indexed columns.

# **MySQL Invisible Index**

**Summary**: in this tutorial, you will learn about MySQL invisible index and the statements to manage the index visibility.

## **Introduction to MySQL invisible index**

The invisible indexes allow you to mark indexes as unavailable for the query optimizer. MySQL maintains the invisible indexes and keeps them up to date when the data in the columns associated with the indexes changes.

By default, indexes are visible. To make them invisible, you have to explicitly declare its visibility at the time of creation, or by using the ALTER TABLE command. MySQL provides us with the VISIBLE and INVISIBLE keywords to maintain index visibility.

To create an invisible index, you the following statement:

CREATE INDEX index\_name

ON table\_name( c1, c2, ...) INVISIBLE;Code language: SQL (Structured Query Language) (sql)

In this syntax:

1. First, you specify the name of the index after the CREATE INDEX clause.
2. Second, you list the table name and the column list which you want to add to the index. The INVISIBLE keyword indicates that the index that you are creating is invisible.

For example, the following statement creates an index on the extension column of the employees table in the sample database and marks it as an invisible index:

CREATE INDEX extension

ON employees(extension) INVISIBLE;

Code language: SQL (Structured Query Language) (sql)

To change the visibility of existing indexes, you use the following statement:

ALTER TABLE table\_name

ALTER INDEX index\_name [VISIBLE | INVISIBLE];Code language: SQL (Structured Query Language) (sql)

For example, to make the extension index visible, you use the following statement:

ALTER TABLE employees

ALTER INDEX extension VISIBLE; Code language: SQL (Structured Query Language) (sql)

You can find the indexes and their visibility by querying the statistics table in the information\_schema database:

SELECT

index\_name,

is\_visible

FROM

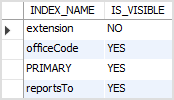
information\_schema.statistics

WHERE

table\_schema = 'classicmodels'

AND table\_name = 'employees';Code language: SQL (Structured Query Language) (sql)

Here is the output:



In addition, you can use the [SHOW INDEXES](https://www.mysqltutorial.org/mysql-index/mysql-show-indexes/) command to display all indexes of a table:

SHOW INDEXES FROM employees;

Code language: SQL (Structured Query Language) (sql)

As mentioned earlier, the query optimizer does not use an invisible index so why do you use the invisible index in the first place?

Practically speaking, invisible indexes have a number of applications. For example, you can make an index invisible to see if it has an impact on the performance and mark the index visible again if it does.

## **MySQL invisible index and primary key**

The index on the [primary key](https://www.mysqltutorial.org/mysql-basics/mysql-primary-key/) column cannot be invisible. If you try to do so, MySQL will issue an error.

In addition, an implicit primary key index also cannot be invisible. When you define a [UNIQUE](https://www.mysqltutorial.org/mysql-unique/) index on a NOT NULL column of a table that does not have a primary key, MySQL implicitly understands that this column is the primary key column and does not allow you to make the index invisible.

Consider the following example.

First, [create a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) with a UNIQUE index on a NOT NULL column:

CREATE TABLE discounts (

discount\_id INT NOT NULL,

name VARCHAR(50) NOT NULL,

valid\_from DATE NOT NULL,

valid\_to DATE NOT NULL,

amount DEC(5 , 2 ) NOT NULL DEFAULT 0,

UNIQUE discount\_id(discount\_id)

);Code language: SQL (Structured Query Language) (sql)

Second, try to make the discount\_id index invisible:

ALTER TABLE discounts

ALTER INDEX discount\_id INVISIBLE;Code language: SQL (Structured Query Language) (sql)

MySQL issued the following error message:

Error Code: 3522. A primary key index cannot be invisible

Code language: SQL (Structured Query Language) (sql)

## **MySQL invisible index system variables**

To control visible indexes used by the query optimizer, MySQL uses the use\_invisible\_indexes flag of the optimizer\_switch system variable. By default, the use\_invisible\_indexes is off:

SELECT @@optimizer\_switch;

Code language: SQL (Structured Query Language) (sql)

In this tutorial, you have learned about the MySQL invisible index, how to create an invisible index, and how to change the visibility of an existing index.

https://www.mysqltutorial.org/mysql-index/mysql-invisible-index/