14.1.14 CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_option]
    [algorithm_option | lock_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

`CREATE INDEX` is mapped to an `ALTER TABLE` statement to create indexes. See Section 14.1.8, “ALTER TABLE Syntax”. `CREATE INDEX` cannot be used to create a `PRIMARY KEY`; use `ALTER TABLE` instead. For more information about indexes, see Section 9.3.1, “How MySQL Uses Indexes”.

Normally, you create all indexes on a table at the time the table itself is created with `CREATE TABLE`. See Section 14.1.18, “CREATE TABLE Syntax”. This guideline is especially important for `InnoDB` tables, where the primary key determines the physical layout of rows in the data file. `CREATE INDEX` enables you to add indexes to existing tables.

A column list of the form `(col1,col2,...)` creates a multiple-column index. Index key values are formed by concatenating the values of the given columns.

For string columns, indexes can be created that use only the leading part of column values, using *`col_name`*`(`*`length`*`)` syntax to specify an index prefix length:

- Prefixes can be specified for `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` column indexes.
- Prefixes *must* be specified for `BLOB` and `TEXT` column indexes.

- Prefix limits are measured in bytes, whereas the prefix length in CREATE TABLE, ALTER TABLE, and CREATE INDEX statements is interpreted as number of characters for nonbinary string types (CHAR, VARCHAR, TEXT) and number of bytes for binary string types (BINARY, VARBINARY, BLOB). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

- For spatial columns, prefix values cannot be given, as described later in this section.

The statement shown here creates an index using the first 10 characters of the name column (assuming that name has a nonbinary string type):

```
CREATE INDEX part_of_name ON customer (name(10));
```

If names in the column usually differ in the first 10 characters, this index should not be much slower than an index created from the entire name column. Also, using column prefixes for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up INSERT operations.

Prefix support and lengths of prefixes (where supported) are storage engine dependent. For example, a prefix can be up to 767 bytes long for InnoDB tables or 3072 bytes if the innodb_large_prefix option is enabled. For MyISAM tables, the prefix limit is 1000 bytes. The NDB storage engine does not support prefixes (see Section 19.1.6.6, “Unsupported or Missing Features in MySQL Cluster”).

MySQL Cluster formerly supported online CREATE INDEX operations using an alternative syntax that is no longer supported. MySQL Cluster now supports online operations using the same ALGORITHM=INPLACE syntax used with the standard MySQL Server. See Section 14.1.8.2, “ALTER TABLE Online Operations in MySQL Cluster”, for more information.

A UNIQUE index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. For all engines, a UNIQUE index permits multiple NULL values for columns that can contain NULL. If you specify a prefix value for a column in a UNIQUE index, the column values must be unique within the prefix.

FULLTEXT indexes are supported only for InnoDB and MyISAM tables and can include only CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 13.9, “Full-Text Search Functions”, for details of operation.

The MyISAM, InnoDB, NDB, and ARCHIVE storage engines support spatial columns such as (POINT and GEOMETRY. (Section 12.5, “Extensions for Spatial Data”, describes the spatial data types.) However, support for spatial column indexing varies among engines. Spatial and nonspatial indexes are available according to the following rules.

Spatial indexes (created using `SPATIAL INDEX`) have these characteristics:

- Available only for `MyISAM` and (as of MySQL 5.7.5) `InnoDB` tables. Specifying `SPATIAL INDEX` for other storage engines results in an error.
- Indexed columns must be `NOT NULL`.
- Column prefix lengths are prohibited. The full width of each column is indexed.

Characteristics of nonspatial indexes (created with `INDEX`, `UNIQUE`, or `PRIMARY KEY`):

- Permitted for any storage engine that supports spatial columns except `ARCHIVE`.
- Columns can be `NULL` unless the index is a primary key.
- For each spatial column in a non-`SPATIAL` index except `POINT` columns, a column prefix length must be specified. (This is the same requirement as for indexed `BLOB` columns.) The prefix length is given in bytes.
- The index type for a non-`SPATIAL` index depends on the storage engine. Currently, B-tree is used.
- You can add an index on a column that can have `NULL` values only for `InnoDB`, `MyISAM`, and `MEMORY` tables.
- You can add an index on a `BLOB` or `TEXT` column only for using the `InnoDB` and `MyISAM` tables.
- When the `innodb_stats_persistent` setting is enabled, run the `ANALYZE TABLE` statement for an `InnoDB` table after creating an index on that table.

As of MySQL 5.7.8, `InnoDB` supports secondary indexes on virtual columns. For more information, see Section 14.1.18.6, “Secondary Indexes and Generated Virtual Columns”.

An ***`index_col_name`*** specification can end with `ASC` or `DESC`. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

Following the index column list, index options can be given. An ***`index_option`*** value can be any of the following:

- `KEY_BLOCK_SIZE [=]` ***`value`***

  For `MyISAM` tables, `KEY_BLOCK_SIZE` optionally specifies the size in bytes to use for index key blocks. The value is treated as a hint; a different size could be used if necessary. A `KEY_BLOCK_SIZE` value specified for an individual index definition overrides a table-level `KEY_BLOCK_SIZE` value.

`KEY_BLOCK_SIZE` is not supported at the index level for `InnoDB` tables. See Section 14.1.18, "CREATE TABLE Syntax".

- ***`index_type`***

  Some storage engines permit you to specify an index type when creating an index. Table 14.1, "Index Types Per Storage Engine" shows the permissible index type values supported by different storage engines. Where multiple index types are listed, the first one is the default when no index type specifier is given. Storage engines not listed in the table do not support an ***`index_type`*** clause in index definitions.

  Table 14.1 Index Types Per Storage Engine

Storage Engine	Permissible Index Types
`InnoDB`	`BTREE`
`MyISAM`	`BTREE`
`MEMORY`/`HEAP`	`HASH`, `BTREE`
`NDB`	`HASH`, `BTREE` (see note in text)

  Example:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index ON lookup (id) USING BTREE;
```

  The ***`index_type`*** clause cannot be used for `FULLTEXT INDEX` or `SPATIAL INDEX` specifications. Full-text index implementation is storage engine dependent. Spatial indexes are implemented as R-tree indexes.

  `BTREE` indexes are implemented by the `NDB` storage engine as T-tree indexes.

> **Note**
>
> For indexes on `NDB` table columns, the `USING` option can be specified only for a unique index or primary key. `USING HASH` prevents the creation of an ordered index; otherwise, creating a unique index or primary key on an `NDB` table automatically results in the creation of both an ordered index and a hash index, each of which indexes the same set of columns.
>
> For unique indexes that include one or more `NULL` columns of an `NDB` table, the hash index can be used only to look up literal values, which means that `IS [NOT] NULL` conditions require a full scan of the table. One workaround is to make sure that a unique index using one or more `NULL` columns on

> such a table is always created in such a way that it includes the ordered index; that is, avoid employing `USING HASH` when creating the index.

If you specify an index type that is not valid for a given storage engine, but another index type is available that the engine can use without affecting query results, the engine uses the available type. The parser recognizes `RTREE` as a type name, but currently this cannot be specified for any storage engine.

> **Note**
>
> Use of the ***`index_type`*** option before the `ON` ***`tbl_name`*** clause is deprecated; support for use of the option in this position will be removed in a future MySQL release. If an ***`index_type`*** option is given in both the earlier and later positions, the final option applies.

`TYPE` ***`type_name`*** is recognized as a synonym for `USING` ***`type_name`***. However, `USING` is the preferred form.

For the storage engines that support an ***`index_type`*** option, Table 14.2, “Storage Engine Index Characteristics” shows some characteristics of index use.

Table 14.2 Storage Engine Index Characteristics

Storage Engine	Index Type	Index Class	Stores NULL Values	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
`InnoDB`	`BTREE`	Primary key	No	No	N/A	N/A
		Unique	Yes	Yes	Index	Index
		Key	Yes	Yes	Index	Index
	Inapplicable	`FULLTEXT`	Yes	Yes	Table	Table
	Inapplicable	`SPATIAL`	No	No	N/A	N/A
`MyISAM`	`BTREE`	Primary key	No	No	N/A	N/A
		Unique	Yes	Yes	Index	Index
		Key	Yes	Yes	Index	Index
	Inapplicable	`FULLTEXT`	Yes	Yes	Table	Table
	Inapplicable	`SPATIAL`	No	No	N/A	N/A
`MEMORY`	`HASH`	Primary key	No	No	N/A	N/A
		Unique	Yes	Yes	Index	Index

		Key	Yes	Yes	Index	Index
	BTREE	Primary	No	No	N/A	N/A
		Unique	Yes	Yes	Index	Index
		Key	Yes	Yes	Index	Index
NDB	BTREE	Primary key	No	No	Index	Index
		Unique	Yes	Yes	Index	Index
		Key	Yes	Yes	Index	Index
	HASH	Primary	No	No	Table (see note 1)	Table (see note 1)
		Unique	Yes	Yes	Table (see note 1)	Table (see note 1)
		Key	Yes	Yes	Table (see note 1)	Table (see note 1)

Table note:

1. If `USING HASH` is specified that prevents creation of an implicit ordered index.

- `WITH PARSER` ***`parser_name`***

  This option can be used only with `FULLTEXT` indexes. It associates a parser plugin with the index if full-text indexing and searching operations need special handling. As of MySQL 5.7.3, `InnoDB` and `MyISAM` support full-text parser plugins. Prior to MySQL 5.7.3, only `MyISAM` supported full-text parser plugins. See Full-Text Parser Plugins and Section 26.2.4.4, “Writing Full-Text Parser Plugins” for more information.

- `COMMENT '`***`string`***`'`

  Index definitions can include an optional comment of up to 1024 characters.

  As of MySQL 5.7.6, the `MERGE_THRESHOLD` for index pages can be configured for individual indexes using the ***`index_option`*** `COMMENT` clause of the `CREATE INDEX` statement. For example:

```
CREATE TABLE t1 (id INT);
CREATE INDEX id_index ON t1 (id) COMMENT 'MERGE_THRESHOLD=40';
```

  If the page-full percentage for an index page falls below the `MERGE_THRESHOLD` value when a row is deleted or when a row is shortened by an update operation, `InnoDB` attempts to merge the index page with a neighboring index page. The default `MERGE_THRESHOLD` value is 50, which is the previously hardcoded value.

  `MERGE_THRESHOLD` can also be defined at the index level and table level using `CREATE TABLE` and `ALTER TABLE` statements. For more information, see Section 15.4.12, “Configuring the Merge

Threshold for Index Pages”.

`ALGORITHM` and `LOCK` clauses may be given to influence the table copying method and level of concurrency for reading and writing the table while its indexes are being modified. They have the same meaning as for the `ALTER TABLE` statement. For more information, see Section 14.1.8, “ALTER TABLE Syntax”