DATA DEFINITION LANGUAGE (DDL) - Syntax

CREATE DATABASE Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
```

`CREATE DATABASE` creates a database with the given name. `CREATE SCHEMA` is a synonym for `CREATE DATABASE`.

`create_specification` A character set is a set of symbols and encodings. A collation is a set of rules for comparing characters in a character set.

Example:

```
CREATE DATABASE IF NOT EXISTS vezbe CHARACTER SET = utf8 COLLATE = utf8_bin;

SHOW CREATE vezbe;

USE vezbe;
```

CREATE TABLE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options]
    [partition_options]
    [IGNORE | REPLACE]
    [AS] query_expression

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    { LIKE old_tbl_name | (LIKE old_tbl_name) }

create_definition:
    col_name column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
      [index_option] ...
  | {INDEX|KEY} [index_name] [index_type] (index_col_name,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
      [index_name] [index_type] (index_col_name,...)
      [index_option] ...
```

```
  | {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] FOREIGN KEY
      [index_name] (index_col_name,...) reference_definition
  | CHECK (expr)

column_definition:
    data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
      [COMMENT 'string']
      [COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
      [STORAGE {DISK|MEMORY|DEFAULT}]
      [reference_definition]
  | data_type [GENERATED ALWAYS] AS (expression)
      [VIRTUAL | STORED] [UNIQUE [KEY]] [COMMENT comment]
      [NOT NULL | NULL] [[PRIMARY] KEY]

data_type:
    BIT[(length)]
  | TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
  | DATE
  | TIME[(fsp)]
  | TIMESTAMP[(fsp)]
  | DATETIME[(fsp)]
  | YEAR
  | CHAR[(length)] [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | VARCHAR(length) [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | BINARY[(length)]
  | VARBINARY(length)
  | TINYBLOB
  | BLOB
  | MEDIUMBLOB
  | LONGBLOB
  | TINYTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | TEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | MEDIUMTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | LONGTEXT [BINARY]
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | ENUM(value1,value2,value3,...)
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | SET(value1,value2,value3,...)
      [CHARACTER SET charset_name] [COLLATE collation_name]
  | JSON
  | spatial_type
```

```
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'

reference_definition:
    REFERENCES tbl_name (index_col_name,...)
      [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
      [ON DELETE reference_option]
      [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION

table_options:
    table_option [[,] table_option] ...

table_option:
    ENGINE [=] engine_name
  | AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | COMPRESSION [=] {'ZLIB'|'LZ4'|'NONE'}
  | CONNECTION [=] 'connect_string'
  | DATA DIRECTORY [=] 'absolute path to directory'
  | DELAY_KEY_WRITE [=] {0 | 1}
  | ENCRYPTION [=] {'Y' | 'N'}
  | INDEX DIRECTORY [=] 'absolute path to directory'
  | INSERT_METHOD [=] { NO | FIRST | LAST }
  | KEY_BLOCK_SIZE [=] value
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PACK_KEYS [=] {0 | 1 | DEFAULT}
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
  | STATS_AUTO_RECALC [=] {DEFAULT|0|1}
  | STATS_PERSISTENT [=] {DEFAULT|0|1}
  | STATS_SAMPLE_PAGES [=] value
  | TABLESPACE tablespace_name [STORAGE {DISK|MEMORY|DEFAULT}]
  | UNION [=] (tbl_name[,tbl_name]...)

partition_options:
    PARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY [ALGORITHM={1|2}] (column_list)
        | RANGE{(expr) | COLUMNS(column_list)}
        | LIST{(expr) | COLUMNS(column_list)} }
    [PARTITIONS num]
```

```
    [SUBPARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY [ALGORITHM={1|2}] (column_list) }
      [SUBPARTITIONS num]
    ]
    [(partition_definition [, partition_definition] ...)]

partition_definition:
    PARTITION partition_name
        [VALUES
            {LESS THAN {(expr | value_list) | MAXVALUE}
            |
            IN (value_list)}]
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'comment_text' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]
        [(subpartition_definition [, subpartition_definition] ...)]

subpartition_definition:
    SUBPARTITION logical_name
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'comment_text' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]

query_expression:
    SELECT ...   (Some valid select or union statement)
```

A **TEMPORARY** table is visible only to the current session, and is dropped automatically when the session is closed. The keywords **IF NOT EXISTS** prevent an error from occurring if the table exists.

data_type represents the data type in a column definition. spatial_type represents a spatial data type. Some attributes do not apply to all data types. **AUTO_INCREMENT** applies only to integer and floating-point types.

Character data types (CHAR, VARCHAR, TEXT) can include **CHARACTER SET** and **COLLATE** attributes to specify the character set and collation for the column. CHARSET is a synonym for CHARACTER SET.

Example:

```
CREATE TABLE t (c CHAR(20) CHARACTER SET latin1 COLLATE latin1_bin);

SHOW CREATE TABLE;
```

The **DEFAULT** clause specifies a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot

set the default for a date column to be the value of a function such as `NOW()` or `CURRENT_DATE`. The exception is that you can specify `CURRENT_TIMESTAMP` as the default for a `TIMESTAMP` or `DATETIME` column.

A comment for a column can be specified with the **`COMMENT`** option, up to 1024 characters long. The comment is displayed by the **`SHOW CREATE TABLE`** and **`SHOW FULL COLUMNS`** statements.

Indexes

`KEY` is normally a synonym for **`INDEX`**. The key attribute **`PRIMARY KEY`** can also be specified as just `KEY` when given in a column definition. This was implemented for compatibility with other database systems.

A **`PRIMARY KEY`** is a unique index where all key columns must be defined as `NOT NULL`. If they are not explicitly declared as `NOT NULL`, MySQL declares them so implicitly (and silently). A table can have only one `PRIMARY KEY`. The name of a `PRIMARY KEY` is always `PRIMARY`, which thus cannot be used as the name for any other kind of index.

If you do not have a `PRIMARY KEY` and an application asks for the `PRIMARY KEY` in your tables, MySQL returns the first **`UNIQUE`** index that has no `NULL` columns as the `PRIMARY KEY`.

In `InnoDB` tables, keep the `PRIMARY KEY` short to minimize storage overhead for secondary indexes. Each secondary index entry contains a copy of the primary key columns for the corresponding row.

A `PRIMARY KEY` can be a multiple-column index. However, you cannot create a multiple-column index using the `PRIMARY KEY` key attribute in a column specification. Doing so only marks that single column as primary. You must use a separate **`PRIMARY KEY(`*`index_col_name, ...`*`)`** clause. You can see index names for a table using **`SHOW INDEX FROM`** ***`tbl_name`***.

Some storage engines permit you to specify an index type when creating an index. The syntax for the `index_type` specifier is **`USING`** ***`type_name`***.

Examples:

```
CREATE TABLE proizvodi (
   maticni_broj TINYINT(3) UNSIGNED NOT NULL,
   ime VARCHAR(50) NOT NULL,
   jed_mere VARCHAR(20) NOT NULL,
   PRIMARY KEY (maticni_broj)
) ENGINE = InnoDB;

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

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.

When you use ORDER BY or GROUP BY on a column in a SELECT, the server sorts values using only the initial number of bytes indicated by the **max_sort_length** system variable. You can create special **FULLTEXT** indexes, which are used for full-text searches. Only the InnoDB and MyISAM storage engines support FULLTEXT indexes. They can be created only from 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.

The **CHECK** clause is **parsed but ignored** by all storage engines.

FOREIGN KEY Constraints

InnoDB and NDB tables support **checking of foreign key constraints**. The columns of the referenced table must always be explicitly named. Both **ON DELETE** and **ON UPDATE** actions on foreign keys are supported.

For other storage engines, MySQL Server parses and ignores the **FOREIGN KEY** and **REFERENCES** syntax in CREATE TABLE statements.

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION
```

Referential Actions

For storage engines supporting foreign keys, MySQL rejects any INSERT or UPDATE operation that attempts to create a foreign key value in a child table if there is no a matching candidate key value in the parent table.

When an UPDATE or DELETE operation affects a key value in the parent table that has matching rows in the child table, the result depends on the *referential action* specified using ON UPDATE and ON DELETE subclauses of the FOREIGN KEY clause. MySQL supports five options regarding the action to be taken, listed here:

- CASCADE: Delete or update the row from the parent table, and automatically delete or update the matching rows in the child table.

Note

Cascaded foreign key actions do not activate triggers.

- `SET NULL`: Delete or update the row from the parent table, and set the foreign key column or columns in the child table to `NULL`.

  If you specify a `SET NULL` action, *make sure that you have not declared the columns in the child table as* `NOT NULL`.

- `RESTRICT`: Rejects the delete or update operation for the parent table.

- `NO ACTION`: A keyword from standard SQL. In MySQL, equivalent to `RESTRICT`. The MySQL Server rejects the delete or update operation for the parent table if there is a related foreign key value in the referenced table. Some database systems have deferred checks, and `NO ACTION` is a deferred check. In MySQL, foreign key constraints are checked immediately, so `NO ACTION` is the same as `RESTRICT`.

- `SET DEFAULT`: This action is recognized by the MySQL parser, but both `InnoDB` and `NDB` reject table definitions containing `ON DELETE SET DEFAULT` or `ON UPDATE SET DEFAULT` clauses.

For an `ON DELETE` or `ON UPDATE` that is not specified, the default action is always `RESTRICT`.

Here is a simple example that relates `parent` and `child` tables through a single-column foreign key:

```
CREATE TABLE parent (
    id INT NOT NULL,
    PRIMARY KEY (id)
) ENGINE=INNODB;

CREATE TABLE child (
    id INT,
    parent_id INT,
    INDEX par_ind (parent_id),
    FOREIGN KEY (parent_id)
        REFERENCES parent(id)
        ON DELETE CASCADE
) ENGINE=INNODB;
```

Example2:

```
CREATE TABLE firme (
   firma TINYINT(3) UNSIGNED NOT NULL,
   naziv_firme VARCHAR(100) NOT NULL,
   mesto VARCHAR(50) NOT NULL,
   adresa VARCHAR(50) NOT NULL,
   telefon VARCHAR(20) NOT NULL,
   PRIMARY KEY (firma)
) ENGINE=INNODB;

CREATE TABLE fakture (
```

```
   sifra_fakture SMALLINT(5) UNSIGNED NOT NULL,
   sifra_firme TINYINT(3) UNSIGNED NOT NULL,
   datum DATE NOT NULL,
   ulaz_izlaz CHAR(1) NOT NULL,
   PRIMARY KEY (sifra_fakture),
   KEY k_sifra_firme (sifra_firme),
   CONSTRAINT fk_fakture_firme FOREIGN KEY (sifra_firme) REFERENCES firme (firma)
   ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE detalji_fakture (
   id MEDIUMINT(8) UNSIGNED NOT NULL AUTO_INCREMENT,
   faktura SMALLINT(5) UNSIGNED NOT NULL,
   red_br TINYINT(3) UNSIGNED NOT NULL,
   proizvod TINYINT(3) UNSIGNED NOT NULL,
   kolicina DECIMAL(8,2) NOT NULL,
   dan_cena DECIMAL(8,2) NOT NULL,
   PRIMARY KEY (id),
   KEY k_proizvod (proizvod),
   KEY k_faktura (faktura),
   CONSTRAINT fk_detalji_fakture_proizvodi FOREIGN KEY (proizvod) REFERENCES
   proizvodi (maticni_broj) ON DELETE CASCADE ON UPDATE CASCADE,
   CONSTRAINT fk_detalji_fakture_fakture FOREIGN KEY (faktura) REFERENCES fakture
   (sifra_fakture) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Adding foreign keys

You can add a new foreign key constraint to an existing table by using `ALTER TABLE`. The syntax relating to foreign keys for this statement is shown here:

```
ALTER TABLE tbl_name
    ADD [CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
```

```
ALTER TABLE proizvodi ADD CONSTRAINT pk_mb PRIMARY KEY (maticni_broj);
```

Dropping Foreign Keys

You can also use `ALTER TABLE` to drop foreign keys, using the syntax shown here:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

To find out the symbol value when you want to drop a foreign key, use a `SHOW CREATE TABLE` statement, as shown here:

Storage Engines

Storage Engine	Description
`InnoDB`	Transaction-safe tables with row locking and foreign keys. The default storage engine for new tables.

Storage Engine	Description
MyISAM	The binary portable storage engine that is primarily used for read-only or read-mostly workloads.
MEMORY	The data for this storage engine is stored only in memory.
CSV	Tables that store rows in comma-separated values format.
ARCHIVE	The archiving storage engine.
EXAMPLE	An example engine.
FEDERATED	Storage engine that accesses remote tables.
HEAP	This is a synonym for MEMORY.
MERGE	A collection of MyISAM tables used as one table. Also known as MRG_MyISAM.
NDB	Clustered, fault-tolerant, memory-based tables, supporting transactions and foreign keys.

When you create a new table, you can specify which storage engine to use by adding an **ENGINE** table option to the CREATE TABLE statement:

```
-- ENGINE=INNODB not needed unless you have set a different
-- default storage engine.
CREATE TABLE t1 (i INT) ENGINE = INNODB;
-- Simple table definitions can be switched from one to another.
CREATE TABLE t2 (i INT) ENGINE = CSV;
CREATE TABLE t3 (i INT) ENGINE = MEMORY;
```

You can specify the default engine by using the `--default-storage-engine` server startup option, or by setting the `default-storage-engine` option in the `my.cnf` configuration file.

CREATE TABLE ... LIKE Syntax

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

LIKE works only for base tables, not for views.

CREATE TABLE ... SELECT Syntax

```
CREATE TABLE new_tbl [AS] SELECT * FROM orig_tbl;
```

```
CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
PRIMARY KEY (a), KEY(b))
ENGINE=MyISAM SELECT b,c FROM test2;
```

Notice that the columns from the SELECT statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
SELECT * FROM foo;
+---+
| n |
```

```
+---+
| 1 |
+---+
```

```
CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

SELECT * FROM bar;
+------+---+
| m    | n |
+------+---+
| NULL | 1 |
+------+---+
1 row in set (0.00 sec)
```

```
CREATE TABLE ... SELECT does not automatically create any indexes for you. This is
done intentionally to make the statement as flexible as possible. If you want to
have indexes in the created table, you should specify these before the SELECT
statement:
```

```
CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

When creating a table with `CREATE TABLE ... SELECT`, make sure to alias any function calls or expressions in the query. If you do not, the `CREATE` statement might fail or result in undesirable column names.

```
CREATE TABLE artists_and_works
  SELECT artist.name, COUNT(work.artist_id) AS number_of_works
  FROM artist LEFT JOIN work ON artist.id = work.artist_id
  GROUP BY artist.id;
```

You can also explicitly specify the data type for a column in the created table:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

CREATE TABLE and Generated Columns

```
The following simple example shows a table that stores the lengths of the sides of
right triangles in the sidea and sideb columns, and computes the length of the
hypotenuse in sidec (the square root of the sums of the squares of the other
sides):
```

```
CREATE TABLE triangle (
  sidea DOUBLE,
  sideb DOUBLE,
  sidec DOUBLE AS (SQRT(sidea * sidea + sideb * sideb))
);

INSERT INTO triangle (sidea, sideb) VALUES(1,1),(3,4),(6,8);

SELECT * FROM triangle;
+-------+-------+--------------------+
| sidea | sideb | sidec              |
```

```
+-------+-------+--------------------+
|     1 |     1 | 1.4142135623730951 |
|     3 |     4 |                  5 |
|     6 |     8 |                 10 |
+-------+-------+--------------------+
```

Generated column definitions have this syntax:

```
col_name data_type [GENERATED ALWAYS] AS (expression)
  [VIRTUAL | STORED] [UNIQUE [KEY]] [COMMENT comment]
  [[NOT] NULL] [[PRIMARY] KEY]
```

`AS (`*expression*`)` indicates that the column is generated and defines the expression used to compute column values. `AS` may be preceded by **`GENERATED ALWAYS`** to make the generated nature of the column more explicit. Constructs that are permitted or prohibited in the expression are discussed later.

The **`VIRTUAL`** or **`STORED`** keyword indicates how column values are stored, which has implications for column use:

- `VIRTUAL`: Column values are not stored, but are evaluated when rows are read, immediately after any `BEFORE` triggers. A virtual column takes no storage.
- `STORED`: Column values are evaluated and stored when rows are inserted or updated.

The default is `VIRTUAL` if neither keyword is specified.

It is permitted to mix `VIRTUAL` and `STORED` columns within a table.

Example:

Suppose that a table `t1` contains `first_name` and `last_name` columns and that applications frequently construct the full name using an expression like this:

```
SELECT CONCAT(first_name,' ',last_name) AS full_name FROM t1;
```

One way to avoid writing out the expression is to create a view `v1` on `t1`, which simplifies applications by enabling them to select `full_name` directly without using an expression:

```
CREATE VIEW v1 AS
SELECT *, CONCAT(first_name,' ',last_name) AS full_name FROM t1;

SELECT full_name FROM v1;
```

A generated column also enables applications to select `full_name` directly without the need to define a view:

```
CREATE TABLE t1 (
  first_name VARCHAR(10),
  last_name VARCHAR(10),
  full_name VARCHAR(255) AS (CONCAT(first_name,' ',last_name))
);

SELECT full_name FROM t1;
```

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. `CREATE INDEX` cannot be used to create a **`PRIMARY KEY`**; use `ALTER TABLE` instead.

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.

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));
```

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.

Table 14.1 Index Types Per Storage Engine

Storage Engine	Permissible Index Types
`InnoDB`	`BTREE`
`MyISAM`	`BTREE`
`MEMORY/HEAP`	`HASH`, `BTREE`

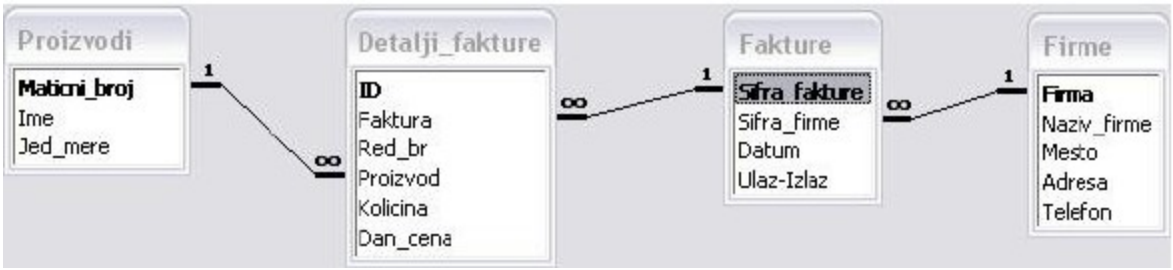
Storage Engine	Permissible Index Types
NDB	HASH, BTREE (see note in text)

Example:

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

ZADACI

1.Neka je kreirana baza podataka poslovanje koja služi za praćenje poslovanja jedne firme koja radi sa određenom grupom proizvoda i sarađuje sa određenim brojem drugih firmi. Baza podataka poslovanje ima četiri tabele i to: priozvodi, firme, fakture i detalji_fakture.
Na sledećoj slici se mogu videti sve tabele sa nazivima kolona i primarnim ključevima (nazivi kolona ispisani polucrnim slovima) i uspostavljene relacije i strani ključevi u odgovarajućim tabelama. Može se videti da su sve uspostavljene relacije tipa jedan-prema više.



Kreirajte tabelu proizvodi

REŠENJE:

Verzija 1
```
CREATE TABLE proizvodi (
    maticni_broj TINYINT(3) UNSIGNED NOT NULL,
    ime VARCHAR(50) NOT NULL,
    jed_mere VARCHAR(20) NOT NULL,
    PRIMARY KEY (maticni_broj)
);
```

Verzija 2
```
CREATE TABLE proizvodi (
    maticni_broj TINYINT(3) UNSIGNED NOT NULL,
    ime VARCHAR(50) NOT NULL,
    jed_mere VARCHAR(20) NOT NULL
);
>>>Query OK, 0 rows affected (0.09 sec)
```

```
ALTER TABLE proizvodi
    ADD CONSTRAINT pk_mb
    PRIMARY KEY (maticni_broj);
>>>Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

2. Kreirajte tabele Fakture i Detalji_fakture koje su povezane preko stranog ključa. Naime, polje Faktura iz tabele Detalji_fakture može uzimati samo one vrednosti koje se pojavljuju u polju Sifra_fakture tabele Fakture. Vodite računa da obezbedite deklarisanje kolona odgovarajućim opcijama PRIMARY KEY, KEY, UNIQUE ili INDEX, kako bi mogao da se automatski formira i indeks. Svi indeksi koji su neophodni najčešće će se kreirati automatski prilikom kreiranja tabele.

REŠENJE:

```
CREATE TABLE firme (
    firma TINYINT(3) UNSIGNED NOT NULL,
    naziv_firme VARCHAR(100) NOT NULL,
    mesto VARCHAR(50) NOT NULL,
    adresa VARCHAR(50) NOT NULL,
    telefon VARCHAR(20) NOT NULL,
    PRIMARY KEY (firma)
);

CREATE TABLE fakture (
    sifra_fakture SMALLINT(5) UNSIGNED NOT NULL,
    sifra_firme TINYINT(3) UNSIGNED NOT NULL,
    datum DATE NOT NULL,
    ulaz_izlaz CHAR(1) NOT NULL,
    PRIMARY KEY (sifra_fakture),
    KEY k_sifra_firme (sifra_firme),
    CONSTRAINT fk_fakture_firme FOREIGN KEY (sifra_firme) REFERENCES firme (firma)
    ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE detalji_fakture (
    id MEDIUMINT(8) UNSIGNED NOT NULL AUTO_INCREMENT,
    faktura SMALLINT(5) UNSIGNED NOT NULL,
    red_br TINYINT(3) UNSIGNED NOT NULL,
    proizvod TINYINT(3) UNSIGNED NOT NULL,
    kolicina DECIMAL(8,2) NOT NULL,
    dan_cena DECIMAL(8,2) NOT NULL,
    PRIMARY KEY (id),
    KEY k_proizvod (proizvod),
    KEY k_faktura (faktura),
    CONSTRAINT fk_detalji_fakture_proizvodi FOREIGN KEY (proizvod) REFERENCES
    proizvodi (maticni_broj) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT fk_detalji_fakture_fakture FOREIGN KEY (faktura) REFERENCES fakture
```

```
    (sifra_fakture) ON DELETE CASCADE ON UPDATE CASCADE
);
```

3. Data je šema relacione baze podataka fubalskog saveza za potrebe evidencije utakmica jedne sezone (pretpostavka je da fudbaleri ne mogu da menjaju tim u kome igraju, u toku sezone):

FUDBALER (SifF, Ime, SifT);
TIM (SifT, Naziv, Mesto);
UTAKMICA (SifU, SifTDomaci, SifTGost, Kolo, Ishod, Godina);
IGRAO (SifF, SifU, PozicijaIgraca);
GOL (SifG, SifU, SifF, RedniBrGola, Minut);
KARTON (SifK, SifU, SifF, Tip, Minut);

Sastaviti SQL skript kojim se formira tabela UTAKMICA, ukoliko je poznato da utakmicu igraju dva različita tima čije se šifre nalaze u tabeli TIM, da ishod utakmice može biti iz skupa vrednosti {X-nerešeno, 1-pobeda domaćih, 2-pobeda gostiju} i da postoji 42 kola u kojima se utakmice igraju. U toku sezone svaki par timova odigra dve utakmice, pri čemu je jedna na domaćem, a druga na gostujućem terenu.

Rešenje:

```
CREATE TABLE UTAKMICA
(
    SifU INT PRIMARY KEY,
    SifTDomaci INT NOT NULL REFERENCES TIM(SifT) ON DELETE CASCADE ON UPDATE
    CASCADE,
    SifTGost INT NOT NULL REFERENCES TIM(SifT) ON DELETE CASCADE ON UPDATE
    CASCADE,
    Kolo INT NOT NULL CHECK (Kolo BETWEEN 1 and 42),
    Ishod CHAR NOT NULL CHECK (Ishod IN ( '1', '2', 'X')),
    Godina INT,
    CHECK (SifTDomaci<> SifTGost),
    UNIQUE (SifTDomaci, SifTGost)
);
```

4. Data je šema relacione baze podataka za potrebe skladišta robe u toku jedne godine:

ROBA(SifR, Naziv, Opis, SifD);
DOBAVLJAC(SifD, Naziv, Adresa);
NABAVKA(SifN, Datum, Kolicina, Cena, SifR);

Sastaviti SQL skript koji formira tabelu NABAVKA ukoliko je poznato da se datum predstavlja kao celobrojna vrednost u opsegu od 1 do 365, i da cena predstavlja jediničnu cenu koja ne sme biti skuplja za više od 10% od cene pri prethodnoj nabavci te robe. Takođe je poznato da jednog datuma može biti najviše jedna nabavka jedne vrste robe.

Rešenje:

```
CREATE TABLE NABAVKA
(
     SifN INT PRIMARY KEY,
    Datum INT NOT NULL CHECK (Datum BETWEEN 1 and 365),
    Cena INT NOT NULL CHECK (Cena>0),
    Kolicina INT NOT NULL CHECK (Kolicina>0),
    SifR INT NOT NULL REFERENCES ROBA(SifR) ON UPDATE CASCADE,
    UNIQUE (SifR, Datum),
    CHECK (NOT EXISTS (SELECT * FROM NABAVKA N1
                    WHERE N1.Cena >= 1.10 *(SELECT N.Cena FROM NABAVKA N2
                                    WHERE N2.SifR=N1.SifR
                                        AND N2.Datum = (SELECT MAX(N3.Datum)
                                                        FROM Nabavka N3
                                                        WHERE N3.SifR=N1.SifR
                                                            AND N3.Datum<N1.Datum
                                                    )
                                    )
                                )
                )
);
```

5. Data je šema relacione baze podataka, za potrebe izračunavanja pomoću matrica:

MATRICA(SifM, Naziv, BrVrsta, BrKolona);
PODACI(SifP, I, J, Vrednost, SifM);

Sastaviti SQL skript koji formira tabelu PODACI ukoliko je poznato da svaka matrica mora imati za svaki element po tačno jednu celobrojnu vrednost.

Rešenje:

```
CREATE TABLE PODACI
(
    SifP INT PRIMARY KEY,
    I INT NOT NULL CHECK (I > 0),
    J INT NOT NULL CHECK (J > 0),
    Vrednost INT NOT NULL,
    SifM INT NOT NULL REFERENCES MATRICA(SifM) ON UPDATE CASCADE,
    UNIQUE (SifM, I, J),
    CHECK (NOT EXISTS (SELECT * FROM PODACI N1
                            GROUP BY SifM
                            HAVING MAX (I) <> COUNT (I)
                                )
                ),
```

```
    CHECK (NOT EXISTS (SELECT * FROM PODACI N1)
                        GROUP BY SifM
                        HAVING MAX (J) <> COUNT (J)
                        )
            )
);
```

6. Data je šema relacione baze podataka veleprodajnog lanca prodavnica:

PRODAVNICA(SifP, Adresa, SifM);
MESTO(SifM, Naziv);
KLIJENT(SifK, Naziv, SifM);
RACUN(SifR, SifK, SifP, SifRa, Datum);
PROIZVOD(SifPr, Naziv, Cena);
STAVKA_RACUNA(SifS, SifR, SifPr, RedniBr, Kolicina, Iznos);
RADNIK(SifRa, Ime, SifP);

Sastaviti SQL skript kojim se formira tabela STAVKA_RACUNA, ukoliko je poznato da se stavke jednog računa moraju unositi redom (sukcesivni RedniBr) i da Iznos mora biti definisan količinom i cenom proizvoda na koji se posmatrana stavka odnosi. Jedan proizvod sena računu ne sme pojavljivati u više stavki.

7. Data je šema relacione deo baze podataka fakulteta:

STUDENT (SifS, Ime, BrIndeksa);
PROFESOR (SifP, Ime, SifO);
ODSEK (SifO, Naziv);
KURS (SifK, Naziv, BrKredita, SifO) UČIONICA (SifU, BrMesta);
PREDUSLOV (SifK, SifKP) POHAĐA(SifS, SifR);
RASPORED (SifR, SifP, SifK, SifU, Termin, Dan, Br.Prijavljenih);

Sastaviti SQL skript kojim se formira tabela RASPORED, ukoliko je poznato da samo jedan profesor može držati po rasporedu predavanje u jednom terminu u jednoj učionici.
Pri tom broj prijavljenih mora biti manji od broja mesta u učionici koja je rasporedu predviđena za taj kurs, a takođe vrednost atributa termin mora biti celobrojna vrednost u opsegu od 1 do 7, a vrednost atributa dan iz skupa vrednosti {Pon, Uto, Sre, Cet, Pet}.

8. Da li je korektan sledeći SQL skript kojim se formira tabela Prodavci?

```
CREATE TABLE Prodavci
(
prod_id CHAR(4) PRIMARY KEY
CHECK (prod_id LIKE '[A-Z][A-Z][A-Z][A-Z]' OR prod_id LIKE'[A -Z][A -Z][0-9][0-9]'),
prod_ime VARCHAR(40),
prod_adresa1 VARCHAR(40),
prod_adresa2 VARCHAR(40),
grad VARCHAR(20),
```

drzava CHAR(2) DEFAULT 'SR',
Postanski_broj CHAR(5) UNIQUE
CHECK (Postanski_broj LIKE '[1-9][0-9][0-9][0-9][0-9]'),
telefon CHAR(12));

Ako je SQL skript korektan, uporedite ga sa rezultatom izvršavanja naredbe
SHOW CREATE TABLE Prodavci;

INSERT INTO Prodavci VALUES('ABCD', 'Pera', 'PeraSt1','PeraSt2','BG',default,'12012','012888888');

9. Data je relacija POSLOVI(sif_radnika, sif_firme, radno_mesto). Ako je ovako određen primarni ključ i ako relacija odgovara realnom modelu jedne firme:
a) Da li radnik može u istoj firmi raditi na više radnih mesta?
b) Da li radnik može raditi u više firmi?
c) Da li firma može imati više radnika zaposlenih na istom radnom mestu?
d) Da li se može upisati novi radnik u relaciju ako nije definisano na kom radnom mestu radi (ako se pri tome podrazumeva da svi atributi u relaciji koji ne pripadaju ključu ne moraju imati zadate vrednosti)

10. Postavite default vrednosti za kolonu grad u tabeli Prodavci i proverite rešenje.
11. Izbacite default vrednosti za kolonu grad u tabeli Prodavci i proverite rešenje.
12. Izbacite kolonu telefon iz tabele Prodavci.

13. Neka su date relacije:

Studenti(BrInd, Ime, Prezime, Adresa, Telefon, Email)
Profesori(IdProf, Ime Prezime, NaucnoZvanje)
Predmeti(IdPredmet, NazivPredmeta)
Ispit(Brind, IdPredmet, IdProf, Ocena, Sala, Datum, Vreme)

Kreirajte SQL iskaze za kreiranje baze i tabela.