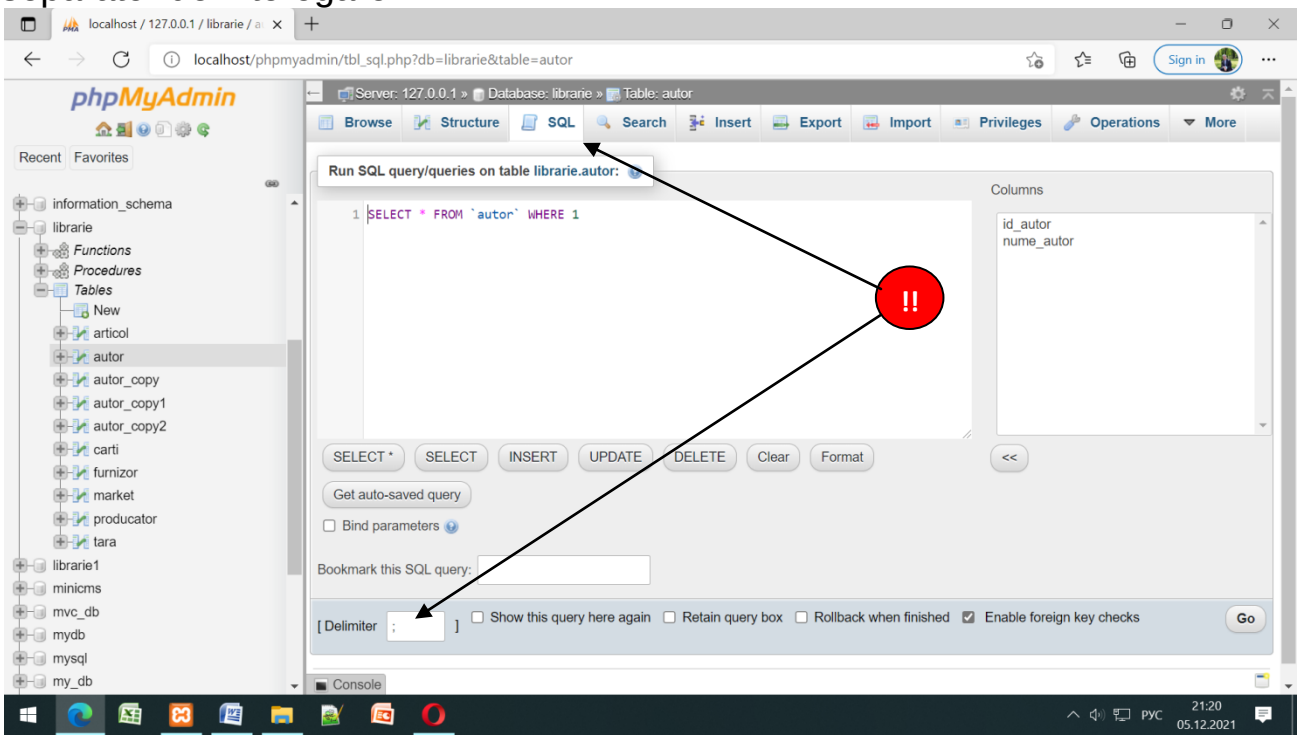
I. DECLANSATOARE/TRIGGERE/ТРИГЕРЫ Create Trigger in MySQL /in PhpMyadmin/

Crearea declanșatorului /un cod/ prin phpMyAdmin

Dacă încercăm să introducem acest cod într-o instrucțiune SQL în phpMyAdmin și să o rulăm așa cum este, aceasta ne va crea unele probleme legate de separatoarele/delimitatoarele de la phpMyAdmin.

Or, este cunoscut că phpMyAdmin folosește implicit **semnul „ ; ”** ca separator de interogare.



Așadar, este IMPORTANT pentru ca declansatoarele sa deruleze in phpMyAdmin, ***în caseta de text*** care se află chiar sub caseta de text unde introduceți interogarea SQL, este necesar de introdus delimitatorul pentru corpul de declansator, daca doriti sa derulati aici corpul trigger-ului.

Este la fel de ușor să se efectueze schimbarea valorii căsuței de text (care este initial **„ ; ”)** cu valoarea „//”.

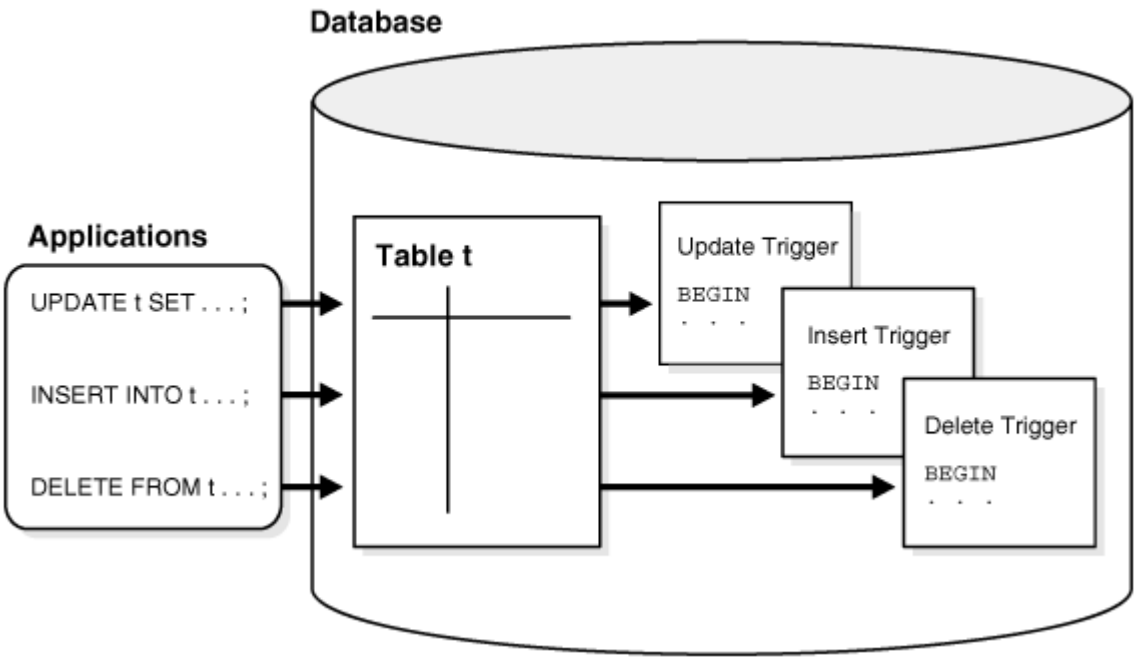
Acum putem rula corpul declansatorului fără probleme, deoarece serverul MySQL înțelege acum că „ **;** ” nu este delimitatorul de interogări SQL, ci este caracterul pentru sfârșitul instrucțiunii, adică „//”.

Notă

Dacă editorul dvs. phpMyAdmin SQL nu vă permite să editați câmpul DELIMITER sau pur și simplu nu vedeți nicăieri acel câmp text, se datorează

faptului că ***versiunea dvs. de phpMyAdmin*** este foarte veche și nu acceptă **modificarea DELIMITER**. Așadar, nu îl puteți modifica utilizând o interogare MySQL, deoarece instrucțiunea DELIMITER nu face parte din setul de instrucțiuni acceptate de serverul MySQL. Este necesar de schimbat versiunea PhpMyadmin si a Serverului MySql.

DECLANSATOARE/TRIGGERE **Create Trigger in MySQL /general/**
http://www.mysqltutorial.org/mysql-triggers.aspx
În MySQL, un declanșator este un program memorat ***invocat automat*** ca răspuns la un eveniment cum ar fi INSERT, UPDATE sau DELETE **care apare în tabelul asociat**.
В MySQL триггер - это сохраненная программа, которая ***автоматически вызывается*** в ответ на такое событие, как INSERT, UPDATE или DELETE, которое появляется в связанной таблице.



De exemplu, puteți defini un declanșator care este invocat automat înainte de introducerea unui nou rând într-un tabel.
MySQL acceptă declanșatoarele care sunt invocate ca răspuns la evenimentul **INSERT** , **UPDATE** sau **DELETE** .

Например, вы можете определить триггер, который запускается автоматически перед вводом новой строки в таблицу. MySQL поддерживает триггеры, которые вызываются в ответ на событие

INSERT , **UPDATE** sau **DELETE** .

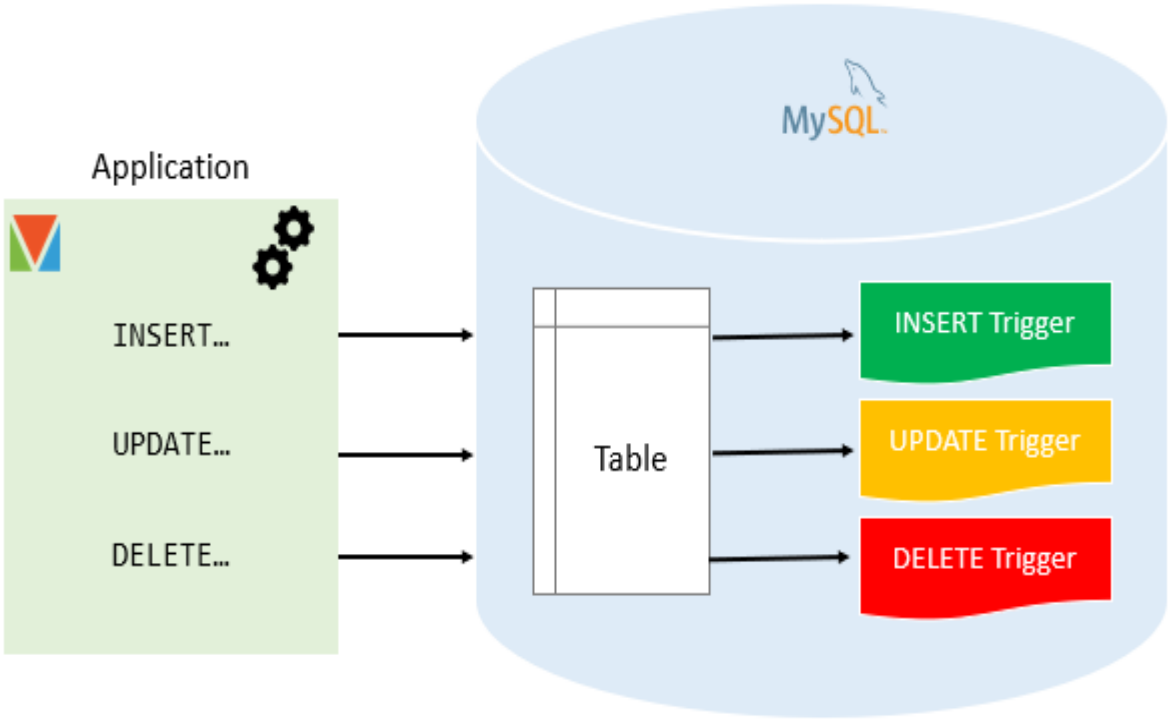
Standardul SQL definește două tipuri de declanșatoare:

1. **declanșatoare la nivel de rând și**
2. **declanșatoare la nivel de declarație /УТВЕРЖДЕНИЯ/.**

- ***Un declanșator la nivel de rând*** **este activat pentru fiecare rând care este introdus, actualizat sau șters /INSERT, UPDATE sau DELETE/**. De exemplu, dacă un tabel are 100 de rânduri introduse, actualizate sau șterse, declanșatorul este invocat automat de 100 de ori pentru cele 100 de rânduri afectate.

Un declanșator la nivel de declarație este ***executat o dată*** pentru ***fiecare tranzacție***, indiferent de rânduri introduse, actualizate sau șterse.

MySQL acceptă numai declanșatoare la nivel de rând. Nu acceptă declanșatorii la nivel de **declarație/comandă**.



Avantajele declanșatoarelor

- Declanșatoarele oferă un alt mod de **a verifica integritatea datelor**.
- Declanșatoarele gestionează erorile in bază de date.
- Declanșatoarele oferă **o** modalitate **alternativă** de a efectua sarcinile programate. Utilizând declanșatoarele, nu trebuie să așteptați să fie difuzate sarcinile programate, deoarece declanșatoarele sunt invocate automat sa deruleze *înainte* sau *după ce* se face o modificare a datelor dintr-un tabel.
- Declanșatoarele **pot fi utile pentru verificarea modificărilor datelor** din tabele.

Преимущества триггеров

1. ***Триггеры предоставляют*** еще один способ проверки целостности данных.
2. ***Триггеры обрабатывают*** ошибки базы данных.
3. ***Триггеры предоставляют альтернативный способ выполнения запланированных задач.*** При использовании триггеров вам не нужно ждать выполнения запланированных задач, поскольку триггеры автоматически вызываются для прокрутки ***до или после изменения данных в таблице***.
4. ***Триггеры могут быть*** полезны для проверки изменений в данных таблицы.

Dezavantaje ale declanșatorilor

- Declanșatoarele pot furniza numai validări extinse, nu toate validările. Pentru validări simple, puteți utiliza restricțiile **`NOT NULL`** , **`UNIQUE`** , **`CHECK`** și **`FOREIGN KEY`** .
- Declanșatoarele pot fi dificil de depanat, deoarece se execută automat în baza de date, ceea ce este posibil să nu fie invizibil pentru aplicațiile client.
- Declanșatoarele **pot crește timpul de lucru al serverului MySQL**.

Недостатки триггеров

1. ***Триггеры могут обеспечивать*** только расширенные проверки, но не все проверки. Для простых проверок вы можете использовать ограничения ***NOT NULL, UNIQUE, CHECK и FOREIGN KEY.***
2. ***Триггеры могут быть трудными*** для устранения неполадок, поскольку они запускаются автоматически в базе данных, что может быть невидимым для клиентских приложений.
3. ***Триггеры могут увеличить время*** работы сервера MySQL.

Gestionarea declanșatoarelor MySQL

- Crearea declanșatori - descrierea pașilor despre cum se poate crea un declanșator în MySQL.
- Drop triggers - cum să eliminăm un declanșator.
- Crearea unui declanșator BEFORE INSERT – se arată cum puteți crea un declanșator **`BEFORE INSERT`** pentru a menține un tabel rezumat dintr-un alt tabel.
- Crearea unui declanșator AFTER INSERT - se arată cum puteți crea un declanșator **`AFTER INSERT`** pentru a insera date într-un tabel după introducerea datelor într-un alt tabel.
- Crearea unui declanșator BEFORE UPDATE - se arată cum puteți crea un declanșator **`BEFORE UPDATE`** care validează datele înainte ca acestea să fie actualizate la tabel.
- Crearea unui declanșator AFTER UPDATE - se arată cum puteți crea un declanșator **`AFTER UPDATE`** pentru a înregistra modificările datelor dintr-un tabel.
- Crearea unui declanșator BEFORE DELETE - se arată cum puteți crea un declanșator **`BEFORE DELETE`** .

- Crearea unui declanșator AFTER DELETE - se arată cum puteți crea un declanșator **AFTER DELETE** .
- Crearea mai multor declanșatoare pentru o tabelă care are același eveniment de declanșare și timp - MySQL 8.0 vă permite să definiți mai multe declanșatoare pentru o tabelă care are același eveniment și declanșator.
- Afișare declanșatoare - listă declanșatoare într-o bază de date, tabelat după tipare specifice.

CREAREA DECLANȘATOARELOR ÎN MYSQL

Instrucțiunea **CREATE TRIGGER** creează un nou declanșator.

Iată **sintaxa de bază** a declarației **CREATE TRIGGER** :

```
CREATE TRIGGER trigger_name
{ BEFORE | AFTER } { INSERT | UPDATE | DELETE }
ON table_name
FOR EACH ROW
trigger_body;
```

În această sintaxă:

- Mai întâi, specificați numele declanșatorului pe care doriți să îl creați după cuvintele cheie **CREATE TRIGGER** . ***Rețineți că numele de declanșare trebuie să fie unic în baza de date.***
- Apoi, ***specificați timpul acțiunii al declanșatorului*** care poate fi, BEFORE sau AFTER ceea ce indică faptul că declanșatorul este invocat **înainte sau după** ce fiecare rând este modificat.
- Apoi, specificați operațiunea care activează declanșatorul, care poate fi INSERT , UPDATE sau DELETE .
- După aceea, specificați ***numele tabelului*** din care face parte declanșatorul după cuvântul cheie ON .
- În cele din urmă, specificați instrucțiunea/(-ile) de executat atunci când declanșatorul se activează. ***Dacă doriți să executați mai multe instrucțiuni, utilizați instrucțiunea compusă {BEGIN …. END}***.

Corpul declanșator poate accesa valorile coloanei afectate de instrucțiunea DML.

Pentru a face distincția/разграничить între valoarea coloanelor **BEFORE și AFTER** – prin care trece DML-ul, utilizăm **modificatorii NEW** și **OLD** .

De exemplu, dacă actualizăm descrierea coloanei, în corpul de declanșare, putem accesa valoarea descrierii înainte de actualizarea **OLD.description** și noua valoare **NEW.description**.

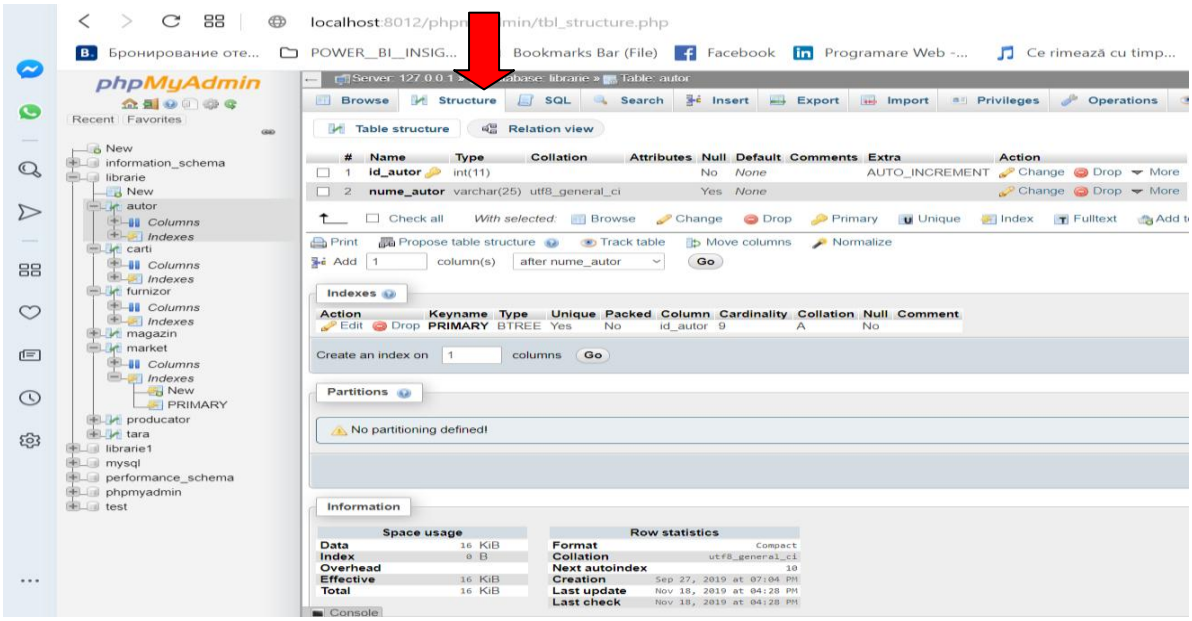
Например, если мы обновим описание столбца в теле триггера, мы сможем получить доступ к значению описания до обновления ***OLD.description*** и новому значению ***NEW.description.***

Următorul tabel ilustrează disponibilitatea modificatorilor OLD și NEW: В следующей таблице показано наличие модификаторов OLD и NEW::

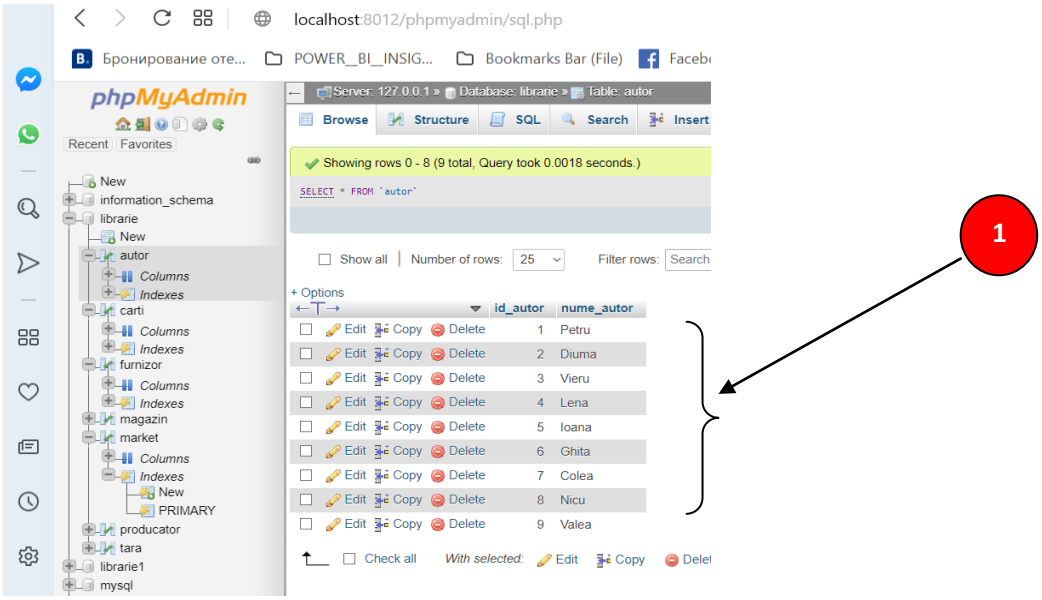
Eveniment declansator	OLD	NEW
INSERT	Nu	da
UPDATE	da	da
DELETE	da	Nu

Exemple de declanșare MySQL

Să creăm un declanșator în MySQL pentru a înregistra modificările tabelului **autor** din BD **librarie**



Sa urmarim ce inscrieri sunt

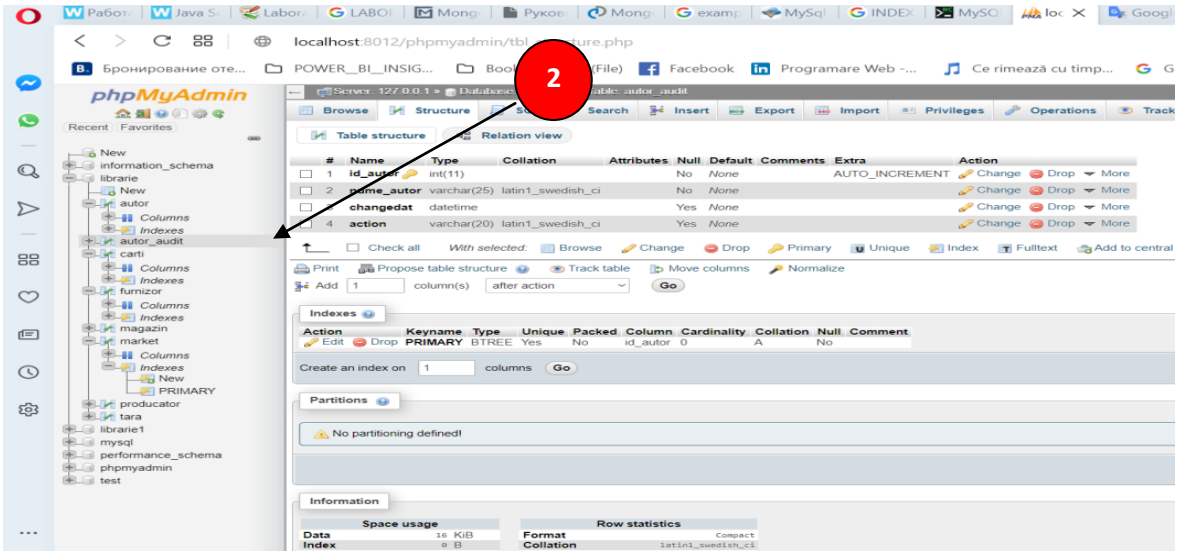


În continuare prin punctul de meniu SQL, **creăm o nouă tabelă** numită ***autor_audit*** pentru a păstra modificările la tabelul ***autor*** :

```
CREATE TABLE autor_audit (
   id_autor INT AUTO_INCREMENT PRIMARY KEY ,
   nume_autor  VARCHAR (25) NOT NULL ,
   changedat DATETIME DEFAULT NULL,
action VARCHAR (50) DEFAULT NULL
);
```

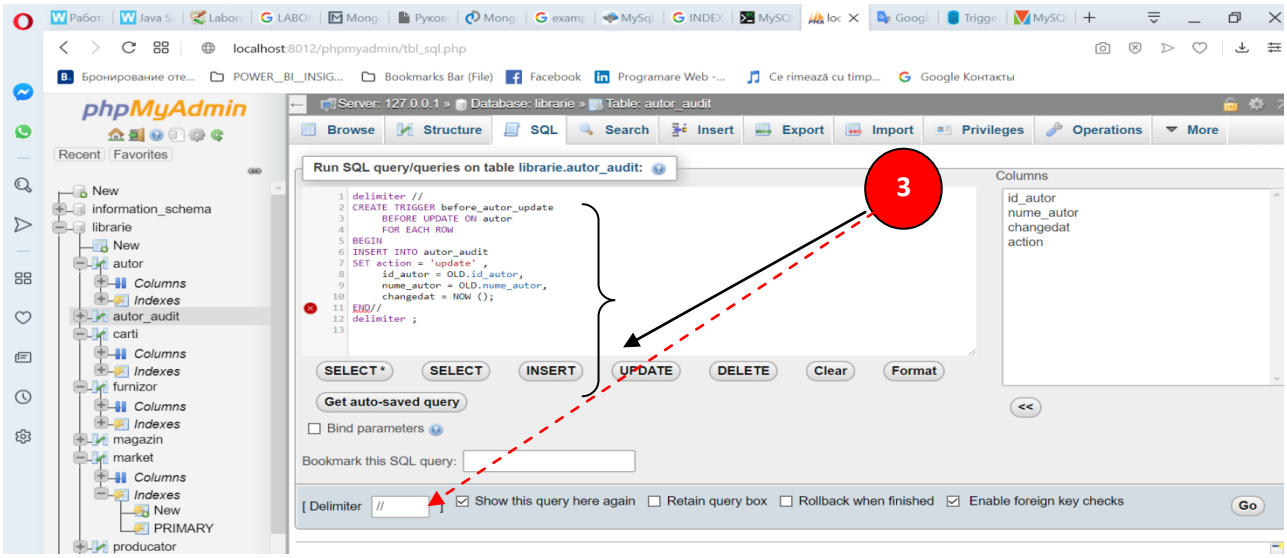
Modificam tabela adaugind un nou cimp pentru a fixa **tipul actiunii**

```
ALTER TABLE autor_audit ADD COLUMN action VARCHAR (20) DEFAULT NULL AFTER changedat ./Apare inca o coloana!!!/
```

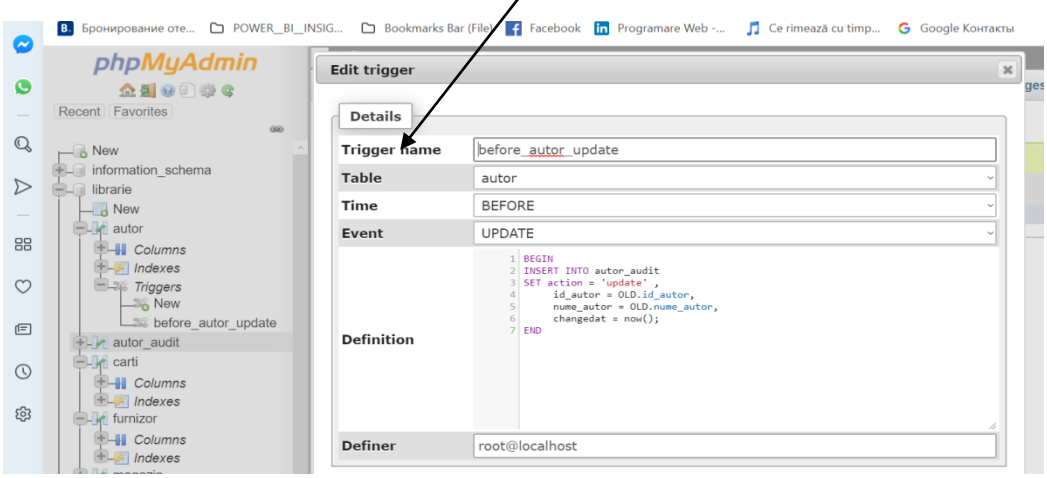


Apoi, creăm un declanșator **BEFORE UPDATE** care este ***lansat/invocat/apelat*** înainte de a face o modificare **în tabelul *autor*.**

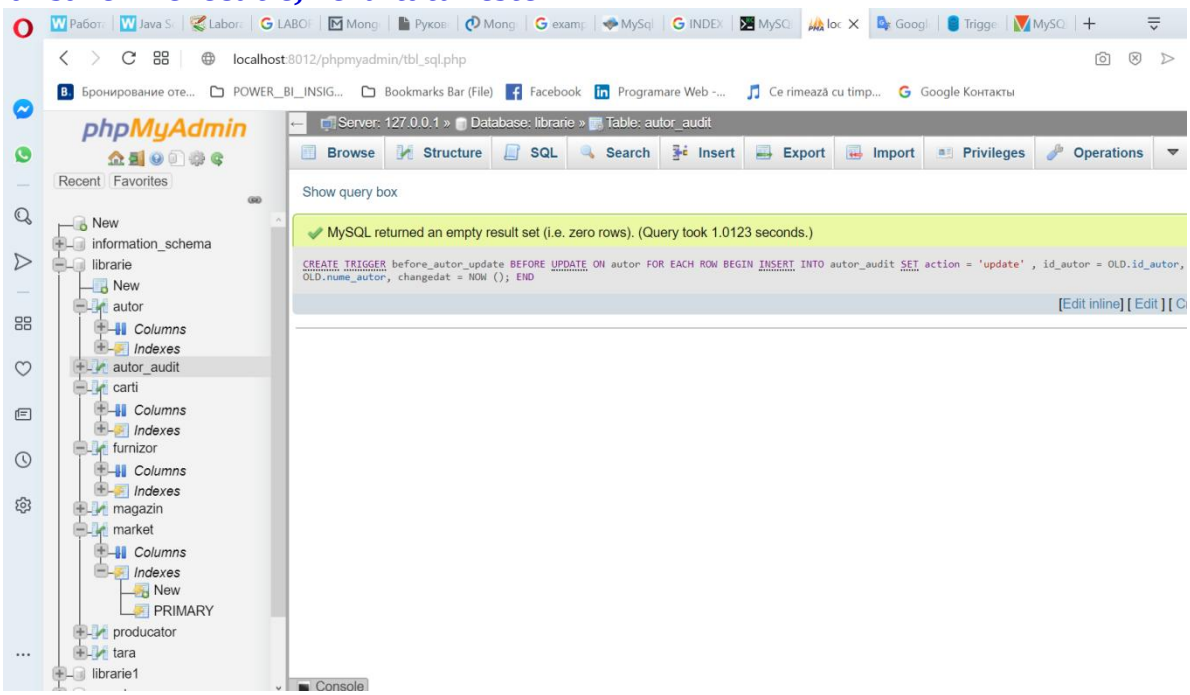
```
DELIMITER //
    CREATE TRIGGER before_autor_update
        BEFORE UPDATE ON autor
        FOR EACH ROW
    BEGIN
    INSERT INTO autor_audit
    SET action = 'update' ,
        id_autor = OLD.id_autor,
        nume_autor = OLD.nume_autor,
        changedat = now();
    END//
DELIMITER ;
```



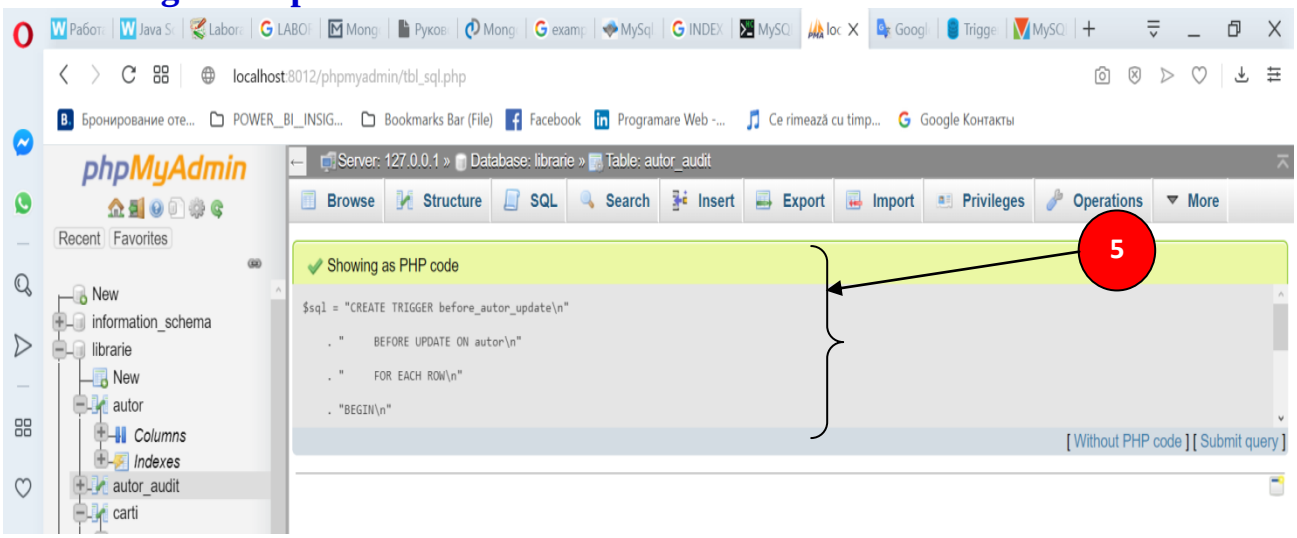
Dacă sunt erori lansam **EDITORUL DE DECLANSATOARE**



Dupa lansare in executie, rezultatul este



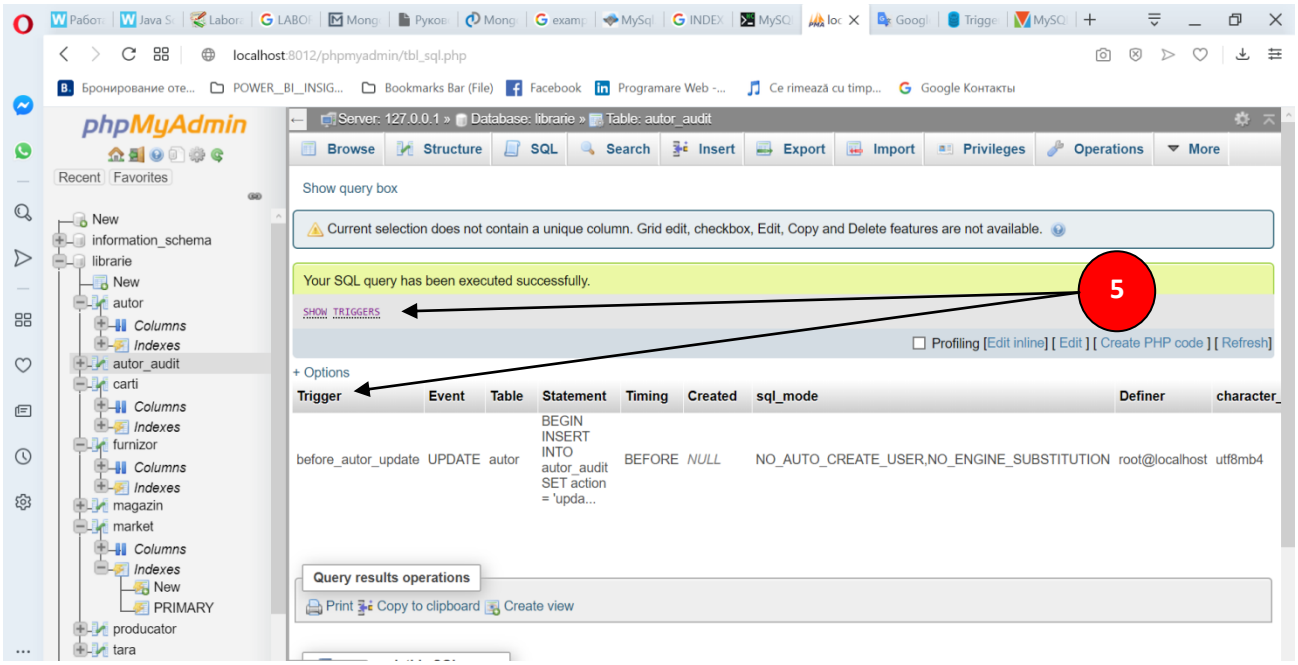
Ori codul generat pentru PHP va fi



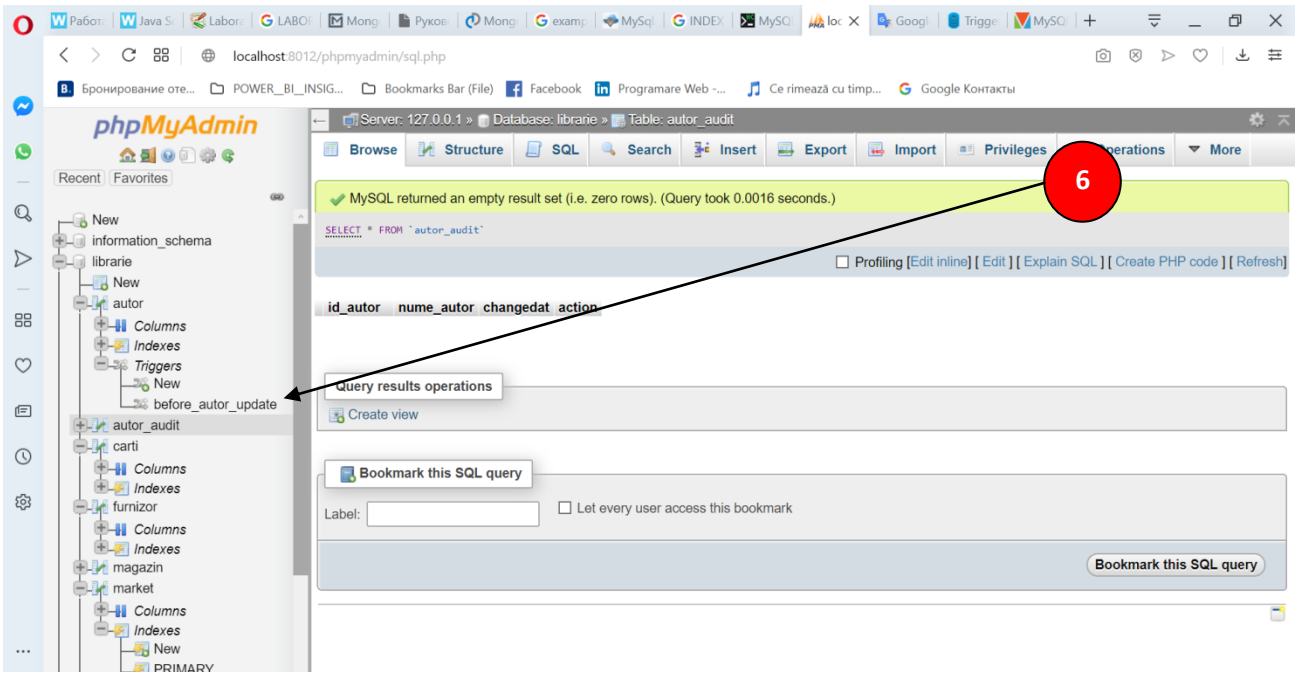
În interiorul corpului declanșatorului, am folosit cuvântul cheie **OLD** pentru a accesa valorile coloanelor ***id_autor, nume_autor*** a **rândului** afectat de declanșator.

Prezentam lista tuturor declanșatoarelor din baza de date curentă folosind comanda

SHOW TRIGGERS

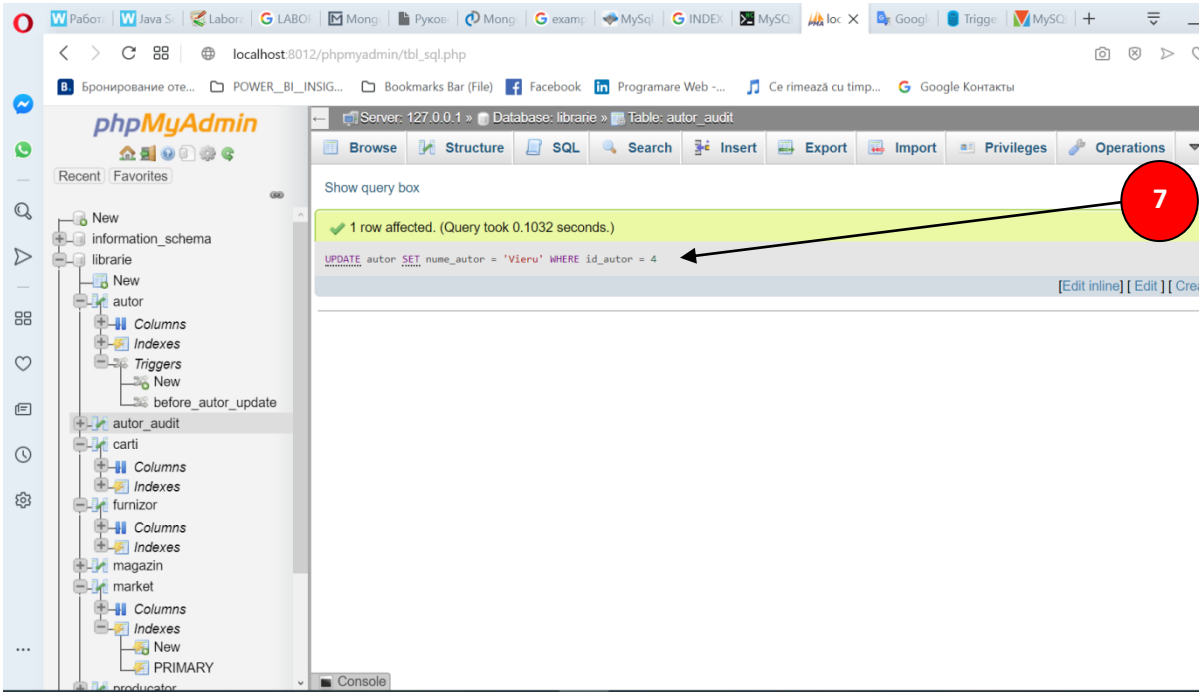


În plus, dacă privim la schemă BD folosind MySQL, in viewer-ul BD putem vedea declanșatorul ***before_autor_update*** așa cum se arată în imaginea de mai jos:



Să actualizam un rând în tabelul autor :

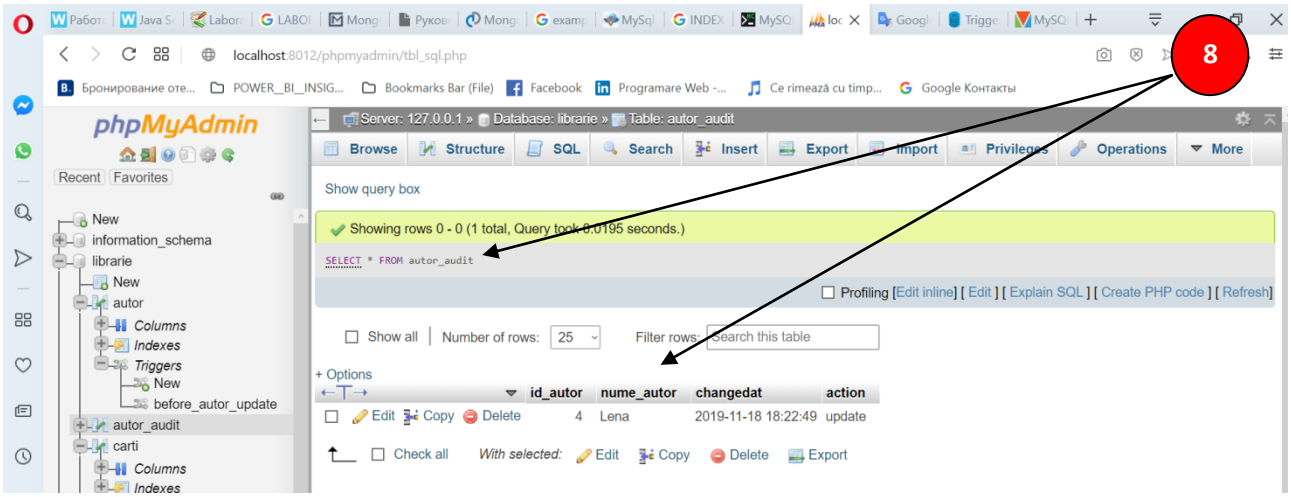
```
UPDATE autor
SET
   Nume_autor = 'Vieru'
WHERE
   Id_autor = 4;
```



În cele din urmă, vom interoga tabelul de la ***autor_audit*** pentru a verifica dacă declanșatorul a fost declanșat prin **declarația UPDATE** :

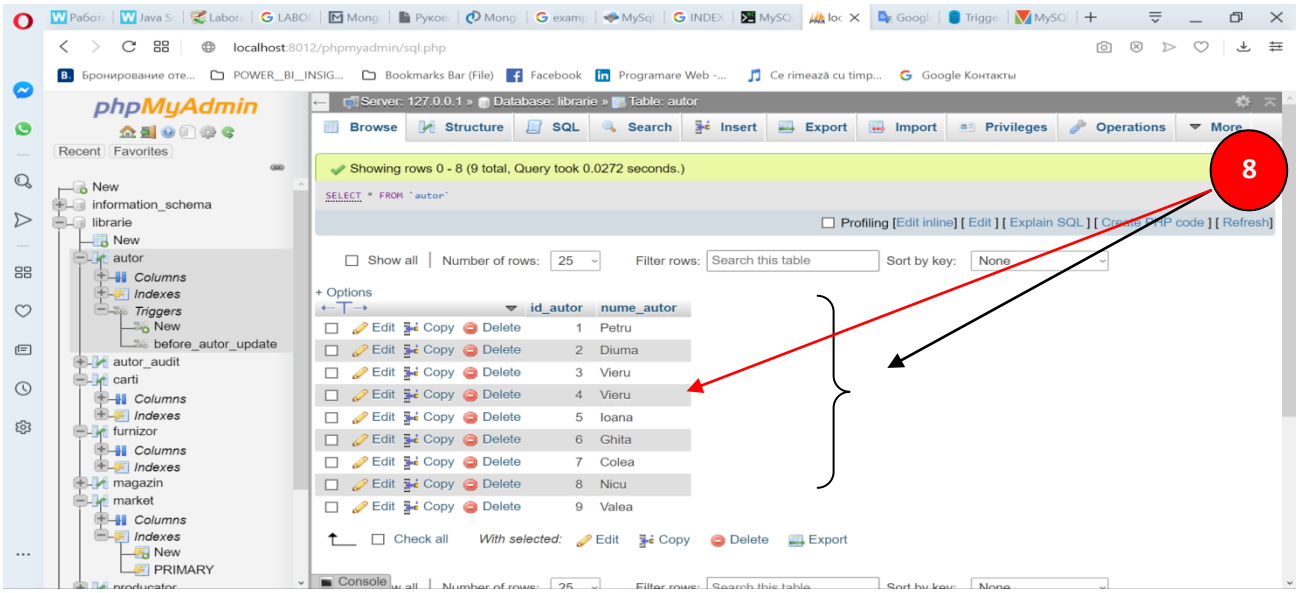
```
SELECT * FROM autor_audit;
```

Următoarea informației demonstrează ieșirea interogării:



După cum vedeți clar din rezultat, declanșatorul a fost invocat automat și a fost introdus un nou rând în tabelul de ***autor_audit*** .

Daca urmăm SELECT pentru tabelul Autor, obținem:



MySQL DROP TRIGGER

Instrucțiunea DROP TRIGGER șterge un declanșator din baza de date.
Iată sintaxa de bază a declarației DROP TRIGGER :

1 DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name;

În această sintaxă:

- ***Mai întâi***, specificați numele declanșatorului pe care doriți să îl renunțați după cuvintele cheie DROP TRIGGER .
- ***În al doilea rând,*** specificați numele schemei din care face parte declanșatorul. Dacă săriți numele schemei, instrucțiunea va renunța la declanșatorul din baza de date curentă.
- ***În al treilea rând,*** utilizați opțiunea IF EXISTS pentru a scoate/elimina condiționat declanșatorul, dacă el există. Clauza IF EXISTS este opțională. Dacă renunțați la un declanșator care nu există fără a utiliza clauza IF EXISTS , MySQL emite o eroare. Cu toate acestea, dacă utilizați clauza IF EXISTS , MySQL emite în schimb o NOTE .

a. DROP TRIGGER necesită privilegiul TRIGGER pentru tabelul asociat declanșatorului.
b. Rețineți că, dacă distrugeti un tabel , MySQL **va distruge automat** la toate declanșatoarele asociate cu tabelul.

Exemplu MySQL DROP TRIGGER

```
DROP TRIGGER  before_autor_update;
```

În cele din urmă, arătați din nou declanșatoarele pentru a verifica eliminarea:

```
SHOW TRIGGERS ;
```

PE ACASA VETI FACE CUNOSTINTA CU

- Crearea unui declanșator ***BEFORE INSERT*** – arătați cum puteți crea un declanșator **`BEFORE INSERT`** pentru a menține un tabel rezumat dintr-un alt tabel.
- Crearea unui declanșator ***AFTER INSERT*** - descrieți cum puteți crea un declanșator **`AFTER INSERT`** pentru a insera date într-un tabel după introducerea datelor într-un alt tabel.
- Crearea unui declanșator ***BEFORE UPDATE*** - arătați cum puteți crea un declanșator **`BEFORE UPDATE`** care validează datele înainte ca acestea să fie actualizate la tabel.
- Crearea unui declanșator ***AFTER UPDATE*** - arătați cum puteți crea un declanșator **`AFTER UPDATE`** pentru a înregistra modificările datelor dintr-un tabel.
- Crearea unui declanșator ***BEFORE DELETE*** - arătați cum puteți crea un declanșator **`BEFORE DELETE`** .
- Crearea unui declanșator ***AFTER DELETE*** - descrieți cum puteți crea un declanșator **`AFTER DELETE`** .
- Crearea mai multor declanșatoare pentru o tabelă care are același eveniment de declanșare și timp - MySQL 8.0 vă permite să definiți mai multe declanșatoare pentru o tabelă care are același eveniment și declanșator.
- Afișare declanșatoare - listă declanșatoare într-o bază de date, tabelat după tipare specifice.

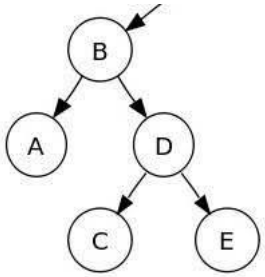
II. INDICȘI IN SQL - MYSQL

Pe degete poate fi explicat după cum urmează: când creăm **un tabel** și adăugăm date in el, tabelul crește și se transformă cu timpul într-o listă secvențială, ordonată după cum au fost adăugate datele.

Când există date puține, **lista este mică și toate cererile** către aceasta sunt executate, aproape imperceptibil. Dar când **numărul de înregistrări din tabel începe să depășească un milion** (în cazuri diferite este diferit, dar ca exemplu, un milion), căutarea datelor. nu este deja atât de rapidă și cu adăugarea din ce în ce a mai multor înregistrări – reduce si mai mult viteza de cautare a datelor.

Acest lucru se datorează faptului că **atunci când căutăm** un fel de înregistrare/inscriere, toate înregistrările sunt scanate până când ajung la cea dorită.
În acest caz, pentru a oprimiza operatiile de căutare/procesare a datelor sunt utilizați.**indecsii**.

Un index este creat pentru un anumit câmp (pot fi utilizate mai multe) prin care se efectuează de obicei o căutare. Când creăm un index, MySql (și orice alt SGBD) trece toate înregistrările din tabel și creează un arbore (cel mai probabil un arbore Binar sau o varietate a lui), în care **în calitate de chei** este ***câmpul selectat***, iar **conținutul**, ***referințele la înregistrările din tabel***.



Atunci când facem următoarea interogare SELECT pe un tabel, pentru câmpul pentru care a fost creat indexul, **MySql** (sau oricare alt SGBD) "știe" că are un indice, care va permite mai rapid parcurgerea, decât sortarea prin toate înregistrările.
Interogarea dvs. va fi redirecționată la acest index și înregistrările ce vor satisface condiției, vor fi găsite mult mai rapid, deoarece o căutare pe arborele construit va fi mult mai rapidă decât o simplă căutare prin toate înregistrările

Și așa deci, **indecsii sunt structuri de date apartinând unui tabel**, si sunt folositi de MySQL ***pentru a mari viteza de cautare***, sau ***de ordonare dupa anumite coloane***. (este vorba de optimizarea cautarilor!)

De exemplu, daca folosim frecvent cautari dupa coloana "**nume**" dintr-un tabel "**persoane**", e optim sa adaugam un index pentru aceasta coloana.

Fara index, MySQL va cauta valoarea respectiva **in fiecare rand** din tabelul nostru, **rand cu rand**, operatie numita ***"full table scan"***, **foarte costisitoare ca timp**.

Cu ajutorul unui index, serverul de baze de date foloseste structura de date aditionala (***ca un fel de tabel suplimentar***) pentru a memora intr-o ordine precisa

valorile coloanei respective, si pentru a ajunge la randul corespunzator valorii cautate mult mai rapid.

Un index poate fi adaugat atat pentru o coloana cat si pentru 2 sau mai multe coloane simultan ("***column indexes***" si "***multiple column indexes***").

Exista urmatoarele tipuri de indecsi in MySQL:

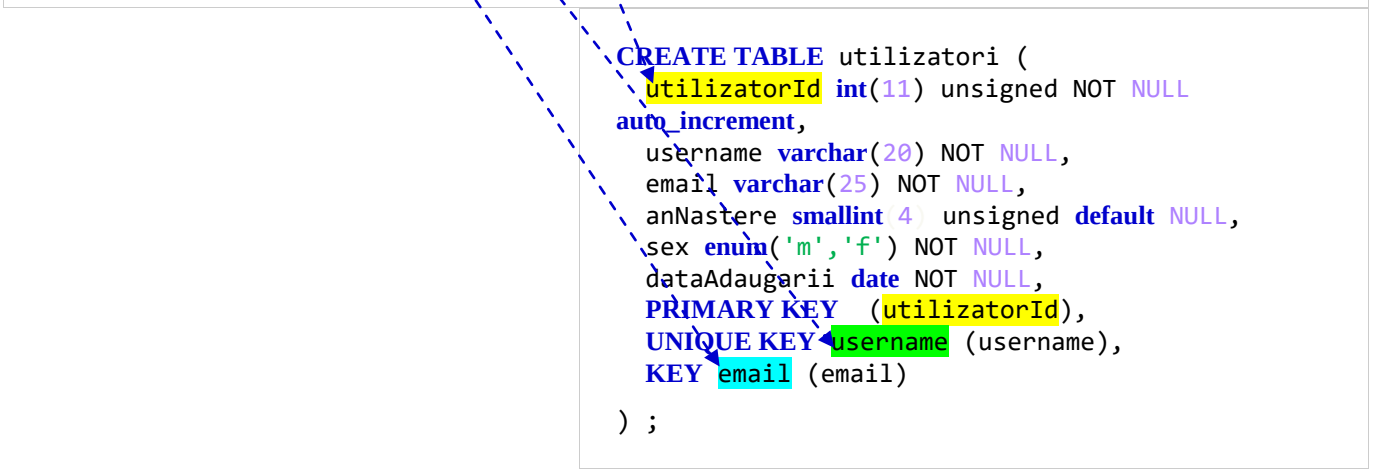
- **Index simplu** - folosind cuvantul cheie **INDEX** sau **KEY** (sinonime)
  Acest index ***nu impune nici o constrangere*** asupra valorilor acestei coloane, ***insa optimizeaza*** (ca orice index) pentru ordonare si cautare.
- **Index unic** - folosind cuvintele **UNIQUE INDEX** sau **UNIQUE KEY**
  Folosind acest index ***impunem ca valorile acestei coloane sa fie unice***. De exemplu, daca avem in tabelul nostru o coloana numita "**email**", un index unic pe aceasta coloana nu va permite existenta unor inregistrari cu aceeasi valoare pentru coloana "**email**". **Exceptia de la aceasta regula** este atunci cand o coloana ***nu are*** atributul ***NOT NULL*** (deci implicit are NULL) si poate aparea valoarea NULL de mai multe ori in cadrul acestei coloane.
- **Cheie primara** - index ce forteaza atat **unicitatea** valorilor acestei coloane cât si prezenta unei valori (nu se accepta valoarea NULL). ***In mod optim***, orice tabel are o coloana cu index-ul **cheie primara**, iar acea coloana este intreg si are in plus atributul **AUTO_INCREMENT**

Exista urmatoarele modalitati prin care se CREEAZA UN INDEX:

- folosind sintaxa **CREATE TABLE** in momentul cand realizam tabelul de exemplu cream cheia primara
- **ALTER TABLE...ADD INDEX**
- folosind **CREATE INDEX** (similar cu ALTER TABLE... ADD INDEX) insa folosind **CREATE INDEX** nu se poate **adauga cheie primara**

EXEMPLE:

- **SE realizeaza tabelul utilizatori, cu 3 indecsi:**
- ***cheie primara* pentru *utilizatorId***
- ***index unic* pentru *username***
- ***index simplu* pentru *email***



```
CREATE TABLE utilizatori (
  utilizatorId int(11) unsigned NOT NULL
auto_increment,
  username varchar(20) NOT NULL,
  email varchar(25) NOT NULL,
  anNastere smallint 4) unsigned default NULL,
  sex enum('m','f') NOT NULL,
  dataAdaugarii date NOT NULL,
  PRIMARY KEY  (utilizatorId),
  UNIQUE KEY username (username),
  KEY email (email)
) ;
```

ALT EXEMPLU

- realizează tabelul ***utilizatori***, doar cu ***indexul cheie primara***
- folosim apoi sintaxa **ALTER TABLE** pentru a adauga un **index unic** pentru ***username*** si unul **simplu pentru *email***

```
CREATE TABLE utilizatori (
  utilizatorId int(11) unsigned  NOT NULL auto_increment,
  username varchar(20) NOT NULL,
  email varchar(25) NOT NULL,
  anNastere smallint (4) unsigned default NULL,
  sex enum('m','f') NOT NULL,
  dataAdaugarii date NOT NULL,
  PRIMARY KEY  (utilizatorId)
) ;
ALTER TABLE utilizatori ADD UNIQUE (username);
ALTER TABLE utilizatori ADD KEY(email);
```

- indecsii astfel adaugati vor optimiza cautari gen:

```
SELECT * FROM utilizatori WHERE email = 'john.suru@gmail.com';
SELECT * FROM utilizatori WHERE username = 'john.suru';
```

Concluzie:

1. ***Succint, indexul,*** **este câmpul** prin care căutarea este optimizată (accelerată).
2. ***Întrucât indexul ocupă spațiu***, este necesar **doar** ***indexarea câmpurilor prin care are loc selecția.***

```
CREATE TABLE utilizatori (
   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   nume VARCHAR(30) NOT NULL,
   prenume VARCHAR(30) NOT NULL,
   email VARCHAR(50),
   reg_date TIMESTAMP
)
```

Aici id – deja este un indecs!!!

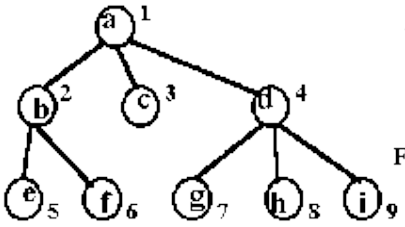
Să admitem că urmează să executăm următoarea interogare (*firstname!*).

```
SELECT * FROM utilizatori WHERE nume = "Ion"
```

Atunci este rezonabil să introducem un indecs după acest cimp

```
CREATE INDEX nume_index ON utilizatori (nume) USING  BTREE;
```

Va fi creată o „***hartă-arbore binar generalizat***” care va facilita găsirea înregistrărilor din lista celor prezente în tabel.



Zona	1	2	3	4	5	6	7	8	9
Cheie	a	b	c	d	e	f	g	h	i
Primul_fiu	2	5	0	7	0	0	0	0	0
Frate_drept	0	3	4	0	6	0	8	9	0

Arborele Binar este un indice arbore echilibrat, este un indice grupat prin frunzele unui arbore echilibrat. ***Este folosit pentru indicsii mari, de fapt este un indice de indici.***

Pe degete, să zicem că indecsii cu o valoare de la **1 la 10** sunt stocati într-o ramură, de la **11 la 20** în alta etc., când vine o solicitare pentru indecsul **35**, se apeleaza la cea de-***a 3-a ramură unde se găseste cel de-al cincilea element.***

Pentru un tabel mic, <1000 de înregistrări, avantajul nu va fi evident! E de ajuns doar să încercăm să îmbinăm alăturând mai mult de (3-4 JOIN-uri) pentru tabele de câmpuri neindexate, și această combinare va reusi să **“omoare”** serverul la un moment dat!

InnoDb si MyISAM:	
	- indecsul primar (PRIMARY KEY) doar **BTREE**
MEMORY	
	- indecsul primar (PRIMARY KEY) prin default **HASH**, se poate allege si BTREE
MySQL 5.7	
	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}

Diferența dintre indicii HASH și BTREE în MySQL

HASH:
- reprezintă rezultatul unei funcții cheie
- nu vede distanța in raport cu cele mai apropiate elemente
- nu poate fi utilizat pentru operații de interval “>” și “<”
- ***compară întreaga cheie***
- liniar la indexare

BTREE:
- indicele indecsilor
- un arbore echilibrat în frunzele căruia este garantat același număr de strămoși
- căutare după intervale, =,>,> =, <, <=, sau operatori BETWEEN
- se poate pentru căutare de utilizat prefixul cheie

- poate fi folosit in comparatii **LIKE**, ***dacă*** argumentul LIKE este un șir constant care ***nu începe*** cu un symbol al sablonului **{% (*), sau _(?)}**

Благодаря такой структуре сложность поиска по btree индексу равна уровням этого дерева – O(log(n)).

EXEMPLU:

SELECT, ce utilizeaza indecsi: **'Patrick*' (in Windows!)**

SELECT * FROM nume_tabel WHERE key_col LIKE 'Patrick%';
- Doar rindurile cu "Patric**k**" <= key_col < 'Patric**l**' vor fi citite

SELECT NU utilizeaza indecsul: **'*Patrick*' (in Windows!)**

SELECT * FROM nume_tabel WHERE key_col LIKE '% Patrick%';
- In acest LIKE valoarea incepe cu simbolul de sablon.

EXEMPLU

Fie dat tabelul **CetOraseRM**, in care avem date despre cetatenii RM inclusiv cu adrese si alte detalii. Evident, vom avea Inscrieri, sute de mii! Sa admitem ca dorim sa efectuam interogarea:

SELECT * FROM CetOraseRM WHERE oras = 'CAHUL'

Aceasta interogare se va derula foarte incet. Pentru a o optimiza este necesar sa adaugam un indecs dupa cum urmeaza:

CREATE INDEX oras_index ON CetOraseRM (oras)

SELECT * FROM `carti` WHERE autor='Petru' 0,0019 sec

CREATE INDEX autor_index ON carti (autor)

SELECT * FROM `carti` WHERE autor='Petru' 0,0017 sec si asta doar la 9 inscrieri!!!

Sa urmarim un exemplu mai solid, care contine multe din notiunile deja studiate.

EXEMPLU

```
CREATE TABLE  'challenges' (
  'ID' int(10) unsigned NOT NULL AUTO_INCREMENT,
  'KIND' enum('1','2','3') NOT NULL DEFAULT '1',
  'TITLE' varchar(255) NOT NULL DEFAULT '',
  'DESCRIPTION' text NOT NULL,
  'DATEAT' datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  'OWNER_ID' int(10) unsigned NOT NULL DEFAULT '0',
  'SOLVEDREPLAY_ID' int(10) unsigned DEFAULT NULL,
  PRIMARY KEY ('ID'),
  KEY 'Index_2' ('OWNER_ID'),
  KEY 'Index_4' ('DATEAT'),
  KEY 'Index_3' ('SOLVEDREPLAY_ID') USING BTREE,
  KEY 'Index_5' ('KIND') USING BTREE,
  CONSTRAINT 'FK_challenges_1' FOREIGN KEY ('OWNER_ID') REFERENCES
'users' ('ID') ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT 'FK_challenges_2' FOREIGN KEY ('SOLVEDREPLAY_ID')
REFERENCES 'uploads' ('ID') ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;
```

EXEMPLU

În MySQL, ELIMINAREA/DISTRUGEREA INDECSULUI este efectuată prin instrucțiuni **DROP INDEX** sau **ALTER TABLE.**

Eliminarea/distrugerea cheilor primare (indicii PRIMARY KEY) poate fi efectuata **doar cu instrucțiunea** ALTER TABLE.

Operatorii de ștergere a indecsului au următoarea sintaxă:

DROP INDEX <index_nume> ON <tbl_nume>

ALTER TABLE <tbl_nume> DROP INDEX <index_nume>

Instrucțiunea de a șterge un index PRIMARY KEY are sintaxa:

ALTER TABLE <tbl_name> DROP PRIMARY KEY

Dacă un astfel de index PRIMARY KEY **nu a fost creat**, iar tabelul are unul sau mai mulți indici UNIQUE, va fi șters primul din ei.

Eliminarea/distrugerea coloanelor dintr-un tabel **afectează indecsii**. Ștergând o coloană dintr-un tabel, eliminam astfel această coloană din indecs. Eliminând toate coloanele indecsate din tabel, ștergem întregul indecs.

DROP INDEX index_name ON table_name

DROP INDEX oras_index on CetOraseRM

Dezavantaje ale inecsilor.

În primul rând, ocupă spațiu pe disc. Și dacă pentru bazele de date mici aceasta este o mică problemă, atunci pentru bazele de date mari, ***indexul poate ocupa mai multă memorie decât întreaga bază de date.***

În al doilea rând, trebuie să fim prudenți. Nu merită să creeăm indecsi pentru toate câmpurile pentru care dorim si putem. Trebuie să fim selectivi în acțiunile noastre. Și dacă indecsii accelerează într-adevăr obtinerea datelor, atunci acest lucru este bun, dar trebuie să cunoastem că ***indecsii micsoreaza viteza de calcul a altor operații cum ar fi – actualizarea UPDATE, inserarea INSERT și ștergerea DELETE datelor***. **Prin urmare, dacă aceste operațiuni sunt foarte des efectuate în baza de date, atunci utilizarea indecsilor ar trebui să fie una limitată**.

Indicșii **FULLTEXT** sunt diferiți de toti cei de mai sus, iar comportamentul lor diferă semnificativ de la SGBD la SGBD. Indicșii FULLTEXT sunt utili doar pentru căutările de text complet efectuate cu clauza **MATCH () / AGAINST (),** spre deosebire de cei trei de mai sus (**KEY** or **INDEX, Unique, Primary**) - care sunt de obicei implementate intern folosind B-arbori (care permit selectarea, sortarea sau intervalele începând de la cei mai din stânga coloana) sau tabele **hash** (care permit selectarea începând de la coloana cei mai din stânga).

În cazul în care celelalte tipuri de index sunt de uz general, un indice ***FULLTEXT este specializat***, întrucât servește un scop restrâns: **este folosit doar pentru o caracteristică de „căutare text completă”.**

https://ylianova.ru/raznoe-2/mysql-indeksy-indeksy-v-mysql.html#_MySQL
В предыдущих статьях я часто упоминал про **индексы в MySQL**. и я обещал, что скоро о них расскажу. Так вот, это время пришло, и сегодня Вы узнаете об **индексах MySQL**, об их назначении и о том, как их создавать.
Индексы используются для ускорения выборки данных из таблиц базы данных. По сути дела, **индекс в MySQL** — это сортировка определённого поля в таблице. То есть если поле сделать индексом, то вся таблица будет отсортирована по этому полю. Почему это выгодно?
Допустим, в нашей таблице находится **1000000** записей. У каждой записи есть уникальный идентификатор **ID**. И, допустим, нам надо вытащить запись с **ID = 530124**. Если нет индекса, то **MySQL** будет поочерёдно перебирать все записи в таблице, пока не найдёт нужную. В худшем случае, он будет вынужден перебрать **1000000** записей. Разумеется, это будет очень долго. А если бы был индекс (то есть поле было бы отсортировано), то выборка записи произошла бы в среднем в **100 000 раз** быстрее. Как видите, выгода очевидна.
Однако, индексы обладают одним существенным изъяном, который не позволяет делать индексом каждое поле таблицы. Фактически, индекс — это ещё одна таблица, но просто с отсортированным соответствующим полем. То есть, делая индекс одного поля, Вы создаёте ещё одну точно такую же таблицу, которая будет занимать дополнительное место на диске.

Ещё один небольшой минус **индексов в MySQL** заключается в том, что запросы на вставку новых записей заставляют проводить сортировку таблицы заново. В результате, вставка новых записей будет происходить несколько дольше обычного. Но не забывайте, что в большиинстве случаев делать это приходится гораздо реже, чем делать выборку, поэтому данный минус не существенен.
Как сделать индекс в MySQL?
Для первичных ключей (**PRIMARY KEY**
) индекс создаётся автоматически, а вот для других полей последовательность действий в **PHPMyAdmin** следующая:

1. Зайти на главную страницу **PHPMyAdmin**.
2. Выбрать из выпадающего списка имя базы данных, где находится требуемая таблица.

3. Кликнуть по имени таблицы, в которую Вы хотите создать индекс.

4. Щёлкнуть на значок «**Молнии**» напротив того поля, для которого Вы хотите создать индекс.

И, напоследок, хочется сделать небольшое резюме, чтобы Вы поняли: «**Когда надо создавать индексы MySQL**«:

- Если по полю очень часто идёт выборка, то его надо делать индексом.
- Если в таблицу очень часто добавляются записи, и при этом выборка происходит редко (такое иногда бывает), то индексы делать не надо.

И ещё кое-что. Если вдруг Вы видите, что Ваши запросы на выборку очень сильно тормозят, то проанализируйте причину этого. Скорее всего, надо просто добавить индекс. В общем, тестируйте, и всё станет понятно