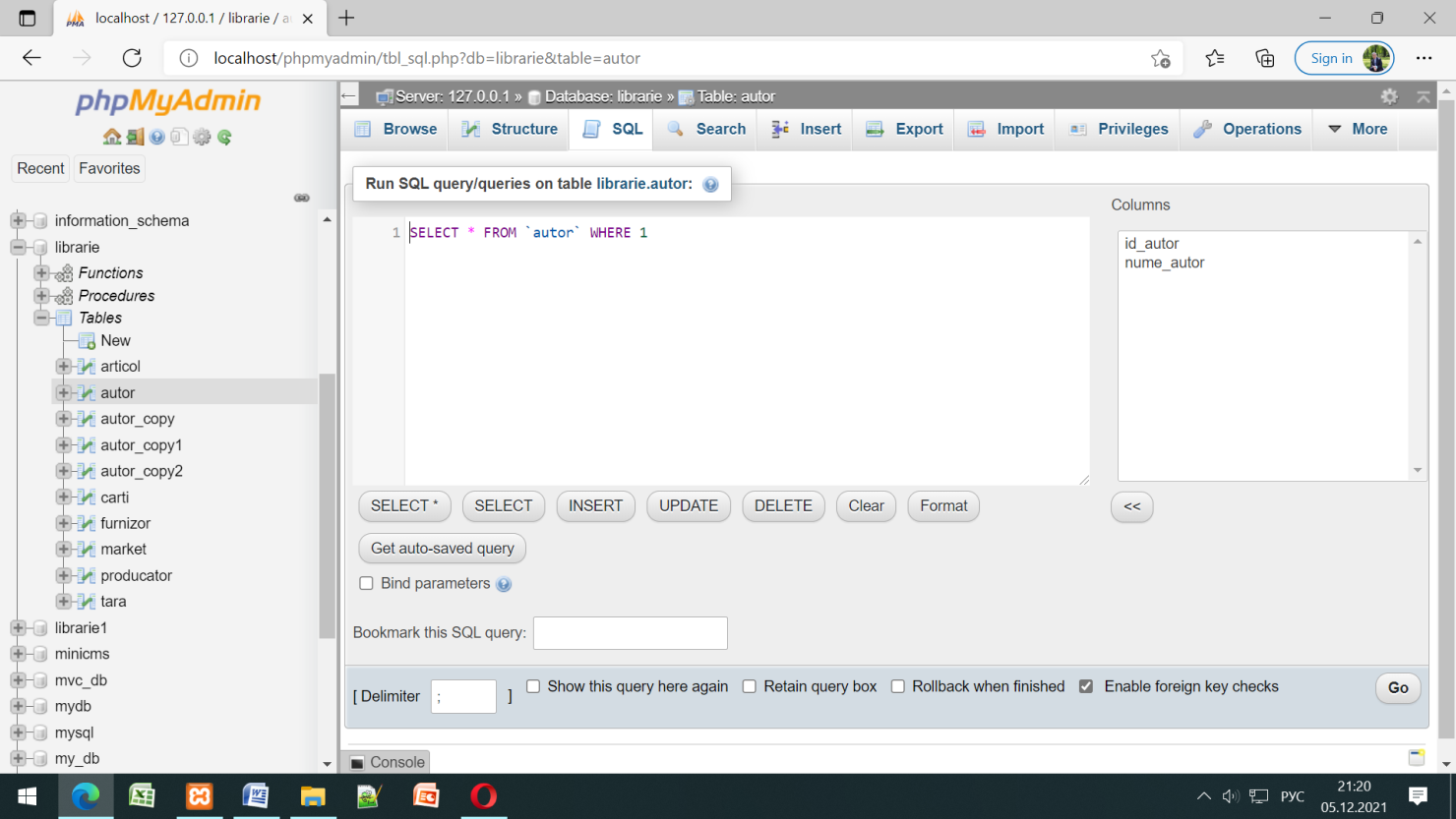
1. **DECLANSATOARE/TRIGGERE / ТРИГЕРЫ**

**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 **IN COMENZILE DE INTEROGARE**.



**3**

**3**

**3**

**1**

***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 /** **разделитель** pentru corpul de declansator, daca doriti sa derulati aici corpul trigger-ului.

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

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 „//” sau „$$”..

**Notă**

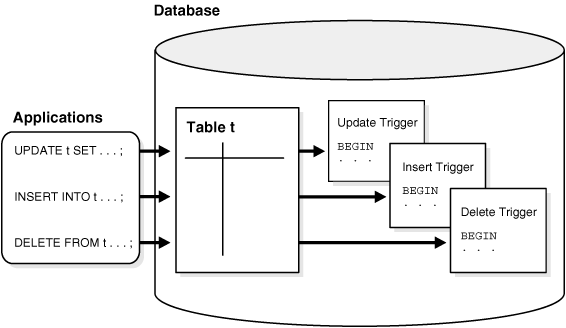
Dacă editorul dvs. phpMyAdmin SQL nu vă permite să editați câmpul **DELIMITER** sau pur și simplu nu vedeți nicăieri acel câmp text, se datorează faptului că ***versiunea dvs. de phpMyAdmin*** este foarte veche și nu acceptă **modificarea DELIMITER**. Așadar, nu îl puteți modifica utilizând o interogare 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 / ТРИГЕРЫ Create Trigger in MySQL /general/**

<http://www.mysqltutorial.org/mysql-triggers.aspx>

***В MySQL триггер*** - это *сохраненная программа*, которая ***автоматически вызывается*** в ответ на такое событие, как ***INSERT, UPDATE или DELETE,*** которое появляется в связанной таблице c этими операциями.

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, adică legat de aceste operații**.



**Например, вы можете определить триггер, который запускается автоматически перед вводом новой строки в таблицу. MySQL поддерживает триггеры, которые вызываются в ответ на такие события как -** **INSERT , UPDATE  sau DELETE** .

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** .

Триггер базы данных - это процедурный код, который автоматически выполняется в ответ на определенные события для конкретной таблицы или представления в базе данных.

***Триггер в основном используется для поддержания целостности информации в базе данных***.

**Например,** когда новая запись (представляющая нового работника) добавляется в таблицу ***сотрудников***, новые записи также должны быть созданы в таблицах ***налогов***, ***отпусков*** и ***зарплат***. Триггеры также можно использовать для регистрации исторических данных, например, ***для отслеживания предыдущих зарплат сотрудников.***

**STANDARDUL SQL DEFINEȘTE DOUĂ TIPURI DE DECLANȘATOARE:**

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

* ***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 câte rânduri sunt ***introduse, actualizate sau șterse***.

**MySQL**

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

**СТАНДАРТ SQL ОПРЕДЕЛЯЕТ ДВА ТИПА ТРИГГЕРОВ:**

**1. триггеры уровня строки и**

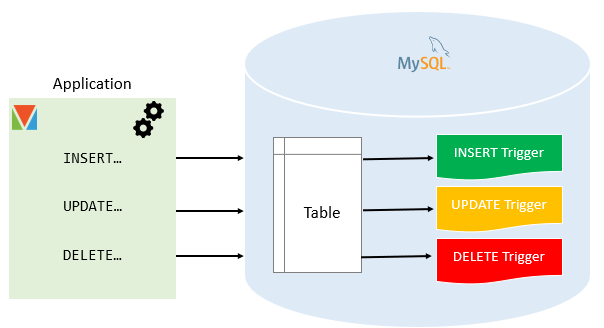
**2. триггеры уровня оператора**

• ***Триггер уровня строки*** **срабатывает для каждой вставляемой, обновляемой или удаляемой строки /INSERT, UPDATE или DELETE/.** Например, если в таблице вставлено, обновлено или удалено 100 строк, триггер автоматически вызывается 100 раз для 100 затронутых строк.

• ***Триггер уровня оператора*** ***выполняется один раз*** ***для каждой транзакции***, независимо от того, сколько строк ***вставлено, обновлено или удалено***.

**MySQL**

* **поддерживает только *триггеры уровня строки*.**
* **не поддерживает Триггеры на уровне операторов/команд,**



**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 нам 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.

**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,](https://translate.googleusercontent.com/translate_c?depth=1&hl=ro&rurl=translate.google.com&sl=en&sp=nmt4&tl=ro&u=http://www.mysqltutorial.org/mysql-triggers/working-mysql-scheduled-event/&xid=17259,15700023,15700043,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhiumjfsJCBCKuwCJ3z2WneLPjEbPQ) 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.

**НЕДОСТАТКИ ТРИГГЕРОВ**

1. ***Триггеры могут обеспечивать*** только ***расширенные проверки***, ***НО не все проверки***. Для простых проверок мы можем использовать ограничения

***NOT NULL, UNIQUE, CHECK и FOREIGN KEY.***

1. ***Триггеры могут быть трудными*** для устранения неполадок, поскольку они запускаются автоматически в базе данных, что может быть невидимым для клиентских приложений.
2. ***Триггеры могут увеличить время работы сервера MySQL***.

**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**.

**Gestionarea declanșatoarelor MySQL**

**Управление триггерами MySQL**

* [Crearea declanșatori](https://translate.googleusercontent.com/translate_c?depth=1&hl=ro&rurl=translate.google.com&sl=en&sp=nmt4&tl=ro&u=http://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx&xid=17259,15700023,15700043,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhiBiNjRGKQYMJhoxFn9oLaOFiOdNA) - 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 /В этом синтаксе:**

• ***Сначала*** указываем ***имя триггера***, который мы хотим создать, после ключевых слов **CREATE TRIGGER.** Обратим внимание, ***что имя триггера должно быть уникальным в базе данных.***

• ***Затем*** ***указываем время срабатывания триггера***, которое может быть ***BEFORE*** или ***AFTER***, что указывает на то, что триггер срабатывает ***до*** или ***после*** ***изменения каждой строки***.

• ***Затем*** указываем ***операцию, запускающую триггер***, которая может быть ***INSERT, UPDATE*** или ***DELETE.***

• ***После*** этого указываем ***имя таблицы***, к которой принадлежит триггер, после ключевого слова ***ON***.

• ***Наконец***, укажите инструкцию (и)/комманды/операторы, которые будут выполняться при срабатывании тригера. Если вы хотите выполнить более одной инструкции, используйте составную инструкцию {***BEGIN…. END}***.

* 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 Data Manipulation Language:**

[INSERT](https://www.geeksforgeeks.org/sql-insert-statement/) : It is used to insert data into a table.

[UPDATE](https://www.geeksforgeeks.org/sql-update-statement/): It is used to update existing data within a table.

[DELETE](https://www.geeksforgeeks.org/sql-delete-statement/) : It is used to delete records from a database table.

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**.

Тело триггера может получить доступ к значениям столбца, затронутого оператором DML. **Data Manipulation Language:**

[INSERT](https://www.geeksforgeeks.org/sql-insert-statement/) : It is used to insert data into a table.

[UPDATE](https://www.geeksforgeeks.org/sql-update-statement/): It is used to update existing data within a table.

[DELETE](https://www.geeksforgeeks.org/sql-delete-statement/) : It is used to delete records from a database table.

Чтобы различать/разграничивать значения столбцов **BEFORE** и **AFTER**, через которые выполняются команды DML, мы используем модификаторы **NEW** и **OLD**.

**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.***

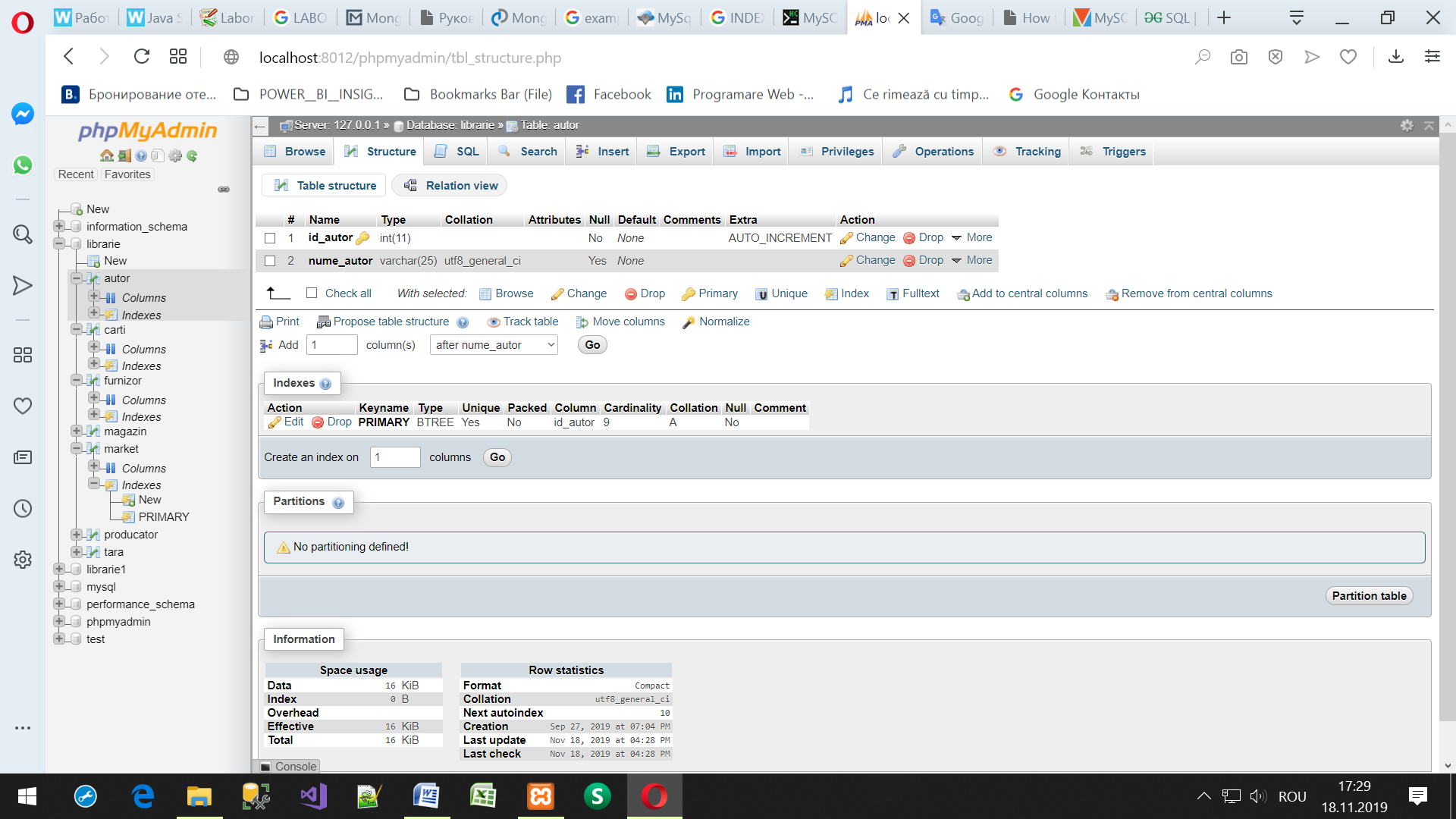
**В следующей таблице показано наличие модификаторов OLD и NEW:**

**Următorul tabel ilustrează disponibilitatea modificatorilor OLD și NEW:**

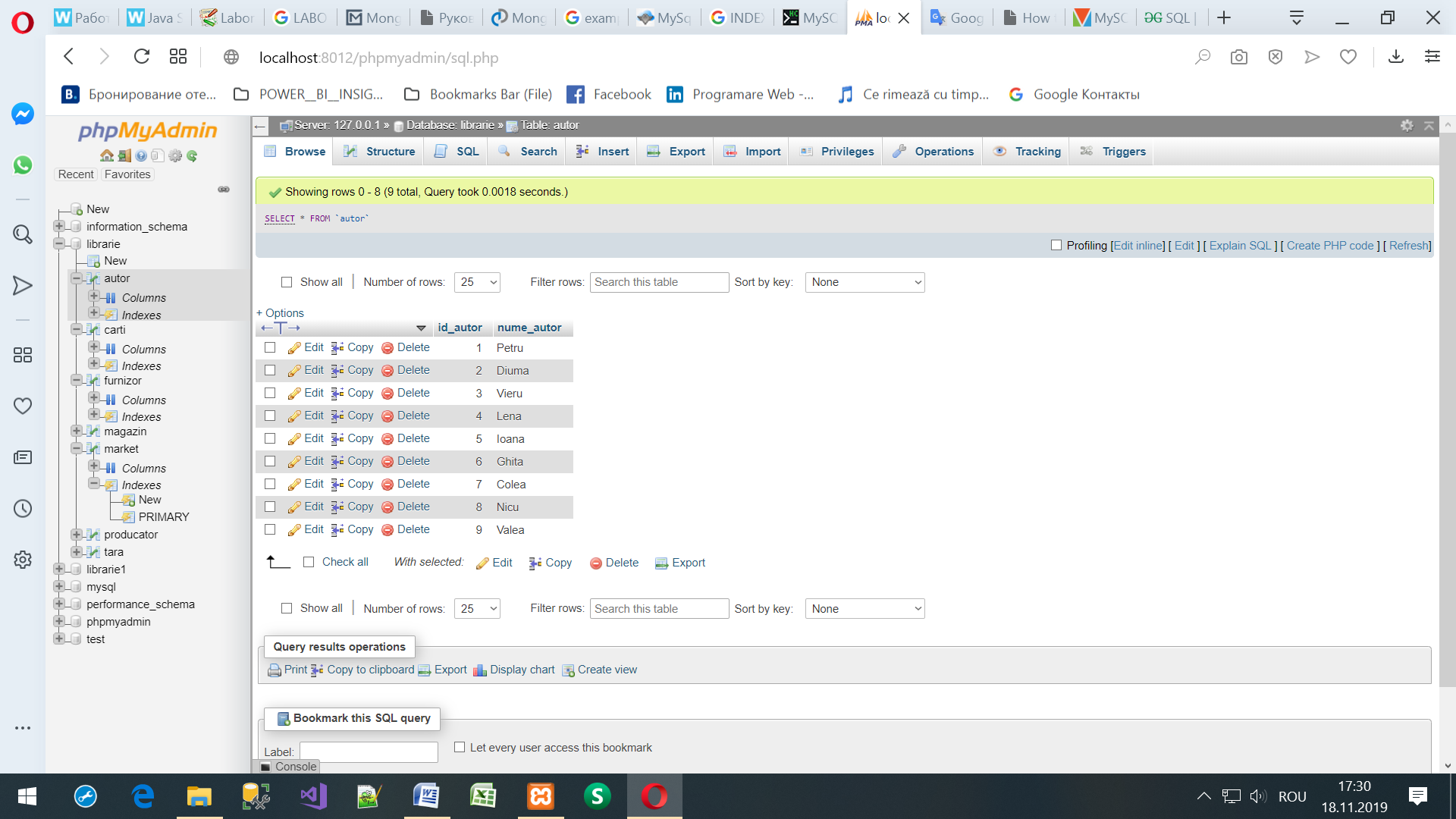
|  |  |  |
| --- | --- | --- |
| **Eveniment declansator**  Триггерное событие | **OLD** | **NEW** |
| INSERT | Nu | da |
| UPDATE | da | da |
| DELETE | da | Nu |

**Exemple de declanșare MySQL**

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



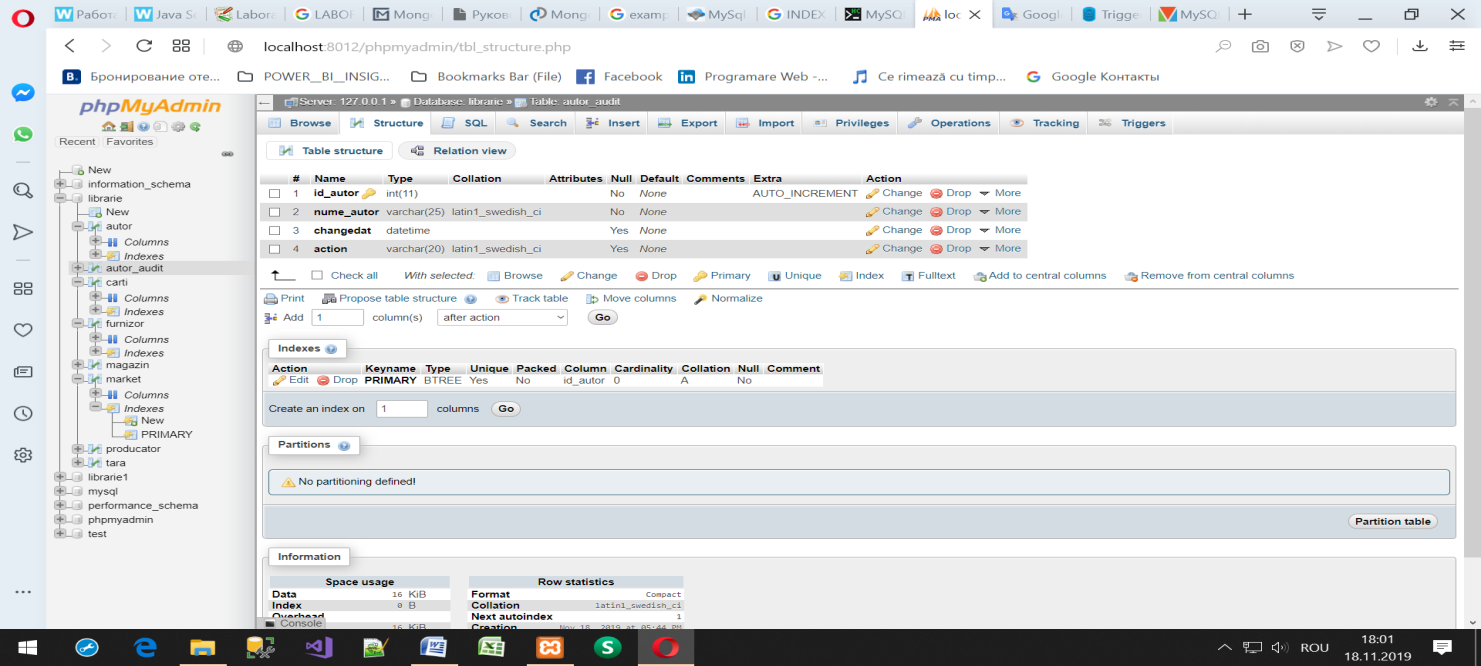
**Sa urmarim ce inscrieri sunt in tabelul Autor**



**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  );  Modificam tabela adaugind un nou cimp pentru a fixa **tipul actiunii** |
| ***action*VARCHAR (20) DEFAULT NULL** |

[**ALTER**](http://localhost:8012/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/alter-table.html)[**TABLE**](http://localhost:8012/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/alter-table.html)**autor\_audit ADD COLUMN action**[**VARCHAR**](http://localhost:8012/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/string-types.html)**(20)**[**DEFAULT**](http://localhost:8012/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/miscellaneous-functions.html#function_default)**NULL AFTER changedat ./Apare inca o coloana!!!/**



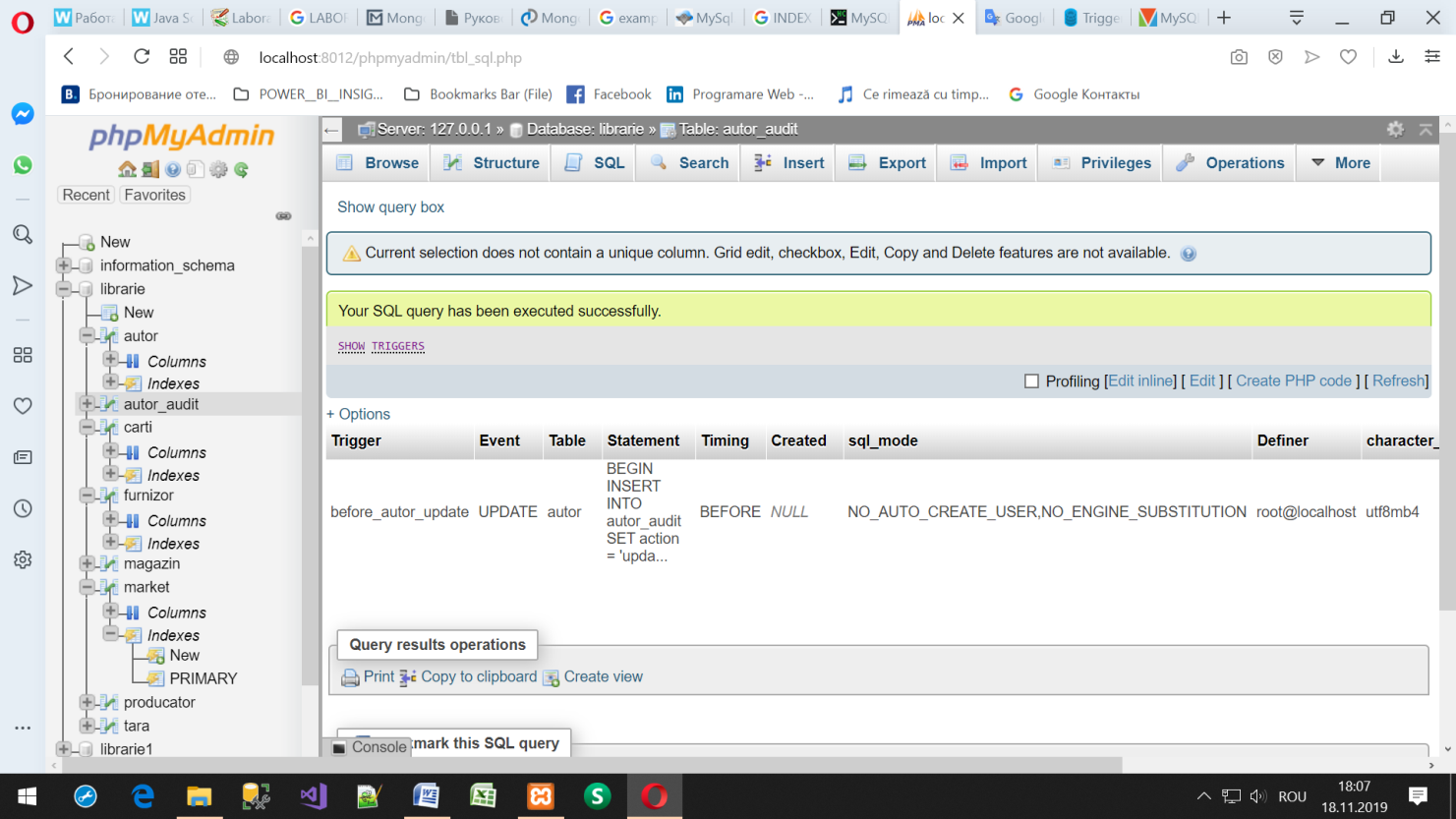
Apoi, creăm un declanșator **BEFORE UPDATE** care este ***lansat/invocat/apelat*** înainte de a face o modificare **în tabelul *autor*.**

|  |
| --- |
| **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** / РЕДАКТОР ТРИГГЕРА    **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.

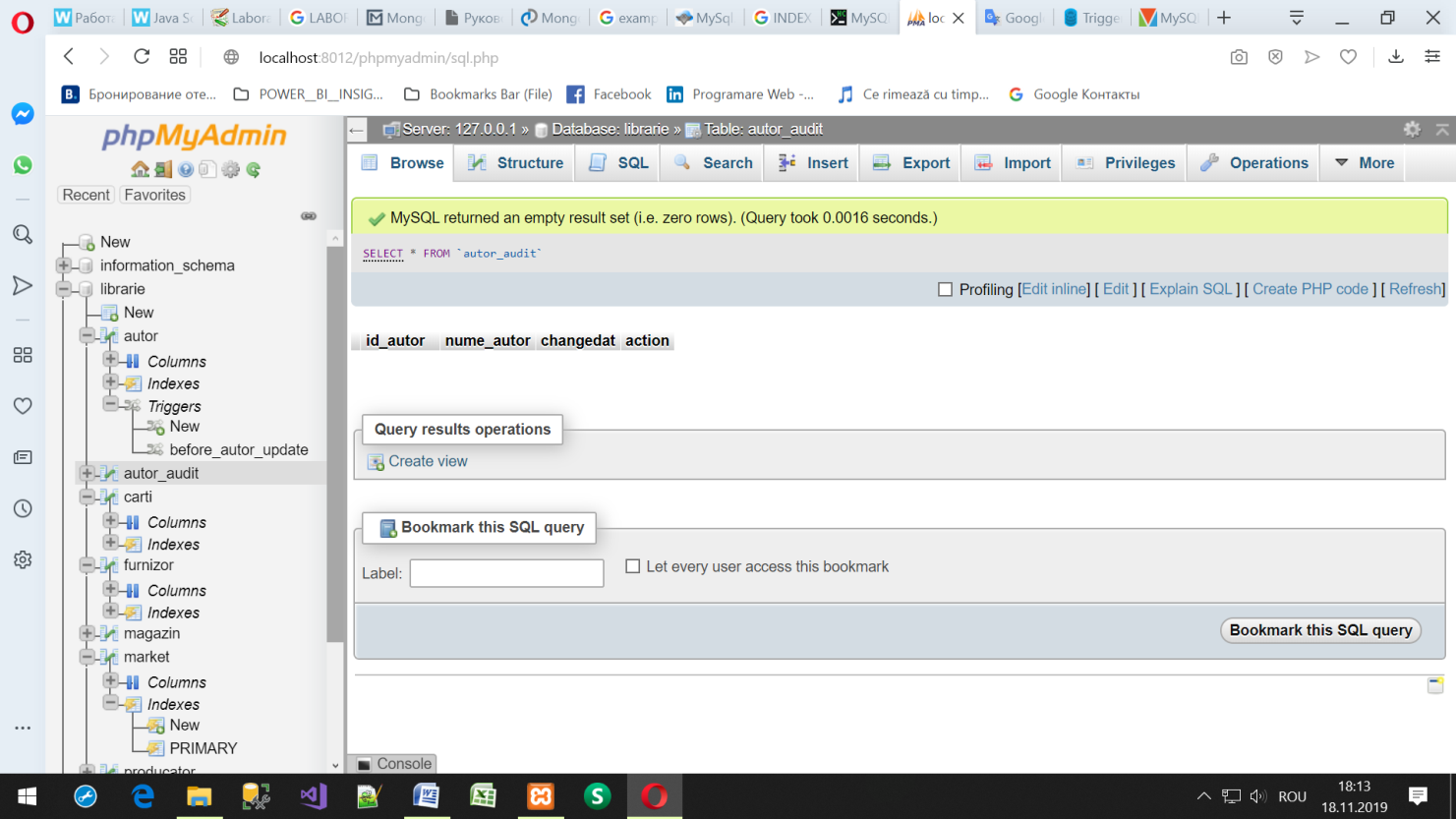
Prezentam lista tuturor declanșatoarelor din baza de date curentă folosind comanda

**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:



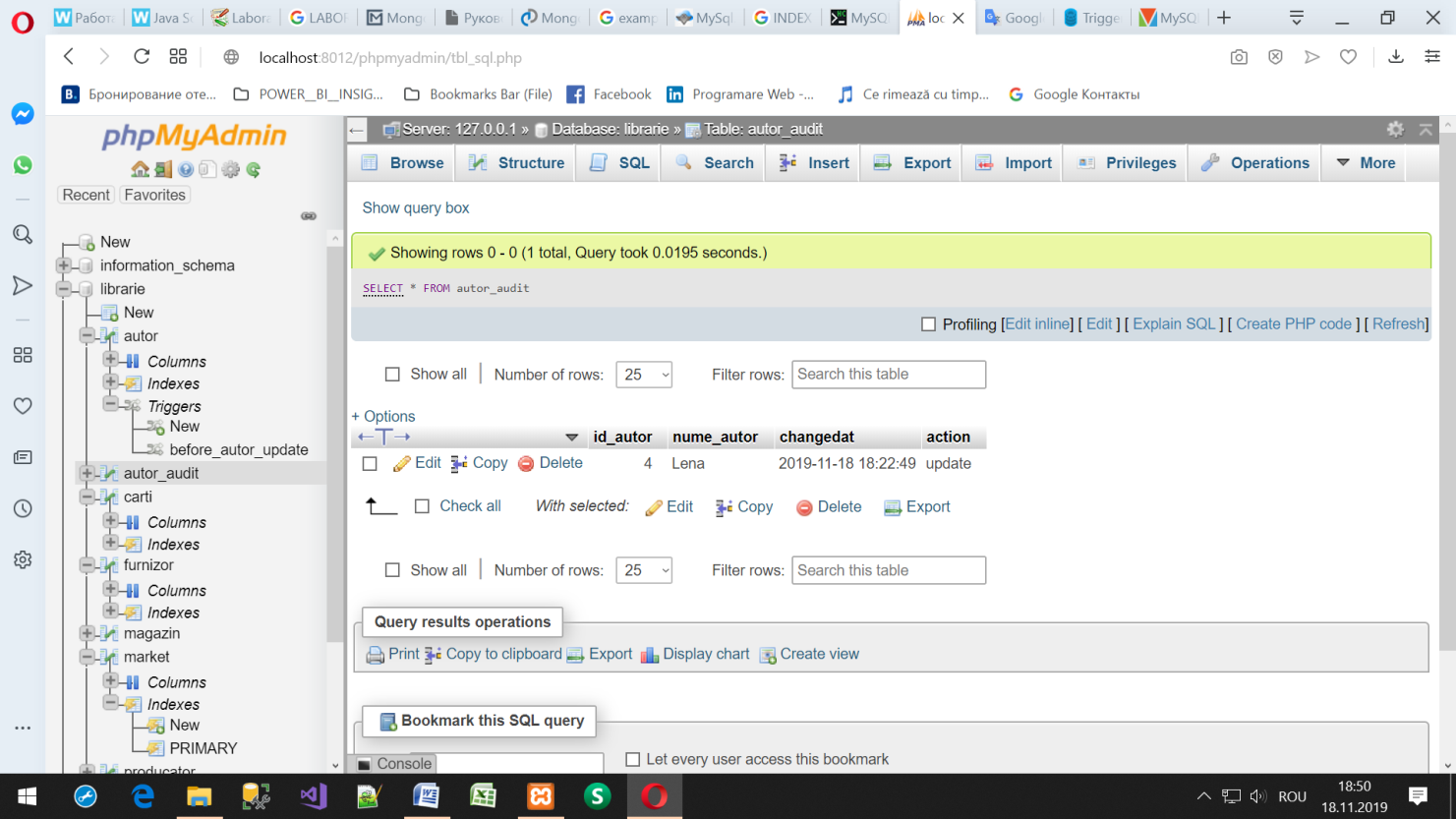
***Să actualizam un rând în tabelul autor  :/ Давйте обновим строку в таблице*** ***autor***:

|  |
| --- |
| 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** :

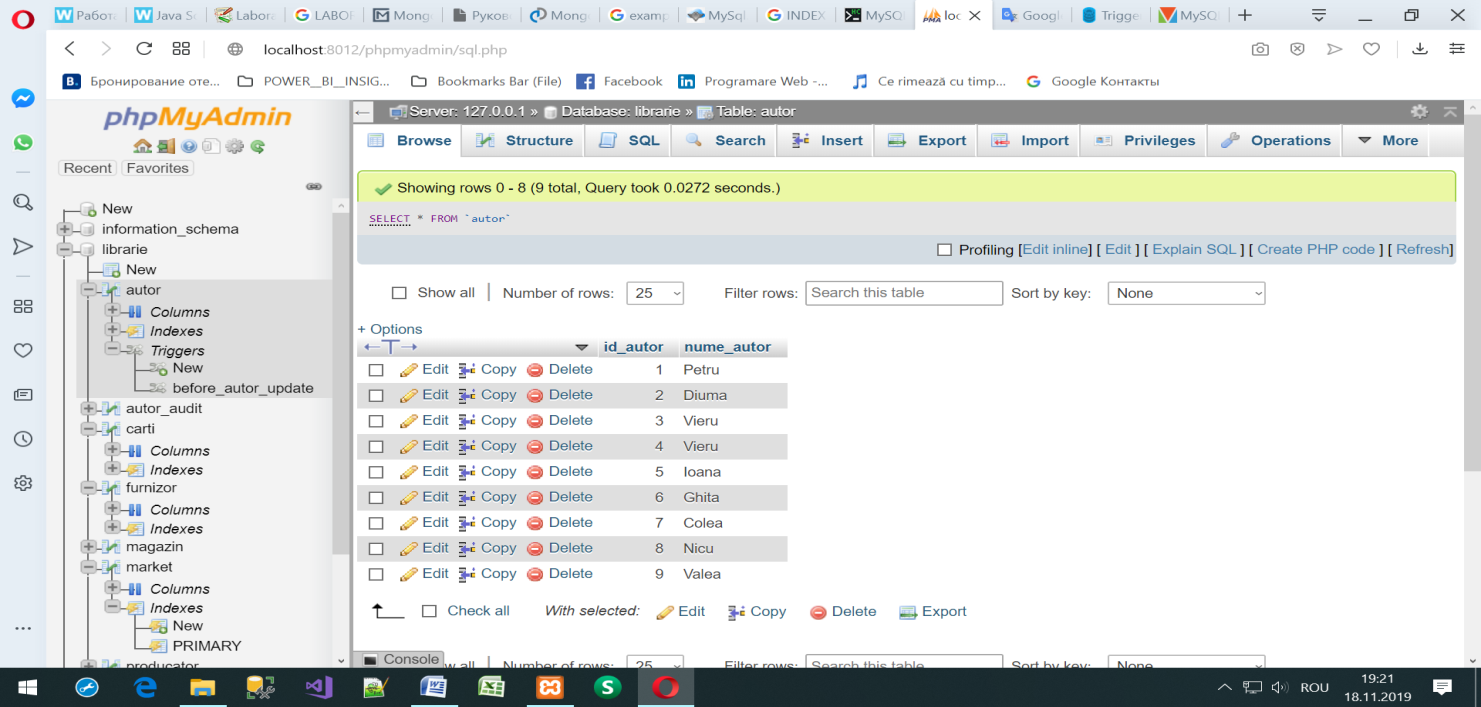
|  |  |
| --- | --- |
|  | **SELECT \* FROM autor\_audit;** |

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-l ștergem*** după cuvintele cheie ***DROP TRIGGER***.
* ***În al doilea rând,*** specificați numele schemei din care face parte declanșatorul. Dacă săriți numele schemei, instrucțiunea ***va sterge*** 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.

* 1. ***DROP TRIGGER*** necesită privilegiul ***TRIGGER*** pentru tabelul asociat declanșatorului.
  2. 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*** – aratați 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*** - aratați cum puteți crea un declanșator **BEFORE UPDATE** care validează datele înainte ca acestea să fie actualizate la tabel.
* Crearea unui declanșator ***AFTER UPDATE*** - aratați cum puteți crea un declanșator **AFTER UPDATE** pentru a înregistra modificările datelor dintr-un tabel.
* Crearea unui declanșator ***BEFORE DELETE*** - aratați 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.

1. **INDICȘI IN SQL – MYSQL**

**На пальцах это можно объяснить следующим образом: когда мы создаем таблицу и добавляем в нее данные, таблица со временем растет и трансформируется в последовательный список, отсортированный по способу добавления данных.**

**Pe degete poate fi explicat după cum urmează: c**ând 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.

**Когда данных мало, список невелик и все запросы к нему выполняются практически незаметно. Но когда количество записей в таблице *начинает превышать один миллион* (в разных случаях он другой, но, например, один миллион), начинается поиск данных и он не так быстр, а с добавлением все большего количества записей - еще больше снижает скорость извлечения данных.**

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.

**Это потому, что, когда мы ищем какую-то запись / запись, все записи сканируются, пока не достигнут желаемой. В этом случае для оптимизации операции поиска / обработки данных используются индексы.**

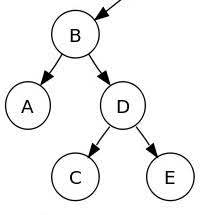
Acest lucru se datorează faptului că **atunci când căutăm** un fel de înregistrare/inscriere, toate înregistrările sunt scanate până când ajung la cea dorită.

În acest caz, pentru a oprimiza operatiile de căutare/procesare a datelor sunt utilizați.**indecsii**.

**Индекс создается для определенного поля (можно использовать несколько полей), по которому обычно выполняется поиск.**

**Когда мы создаем индекс, MySql (и любая другая СУБД) передает все записи в таблице и создает дерево (скорее всего, двоичное дерево или его разновидность), в котором выбранное поле является ключом, а содержимое - ссылки на записи в таблице.**

**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***.



**Когда мы выполняем следующий запрос SELECT для таблицы, для поля, для которого был создан индекс, MySql (или любая другая СУБД) «знает», что у нее есть индекс, который позволит быстрее просматривать, чем сортировать все записи.**

***Ваш запрос будет перенаправлен на этот индекс, и записи, удовлетворяющие условию, будут найдены намного быстрее, потому что поиск по встроенному дереву будет намного быстрее, чем простой поиск по всем записям.***

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.***

**Итак, индексы - это структуры данных, которые принадлежат таблице и *используются MySQL для увеличения скорости поиска или сортировки по определенным столбцам*. (Это про поисковую оптимизацию!)**

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

**Например, если мы часто используем поиск столбца «nume/имя» в таблице «persoane», лучше всего добавить индекс для этого столбца.**

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

**Без индекса, MySQL будет искать это значение в каждой строке нашей таблицы, строка за строкой, операция, называемая «полное сканирование таблицы» *"full table scan"*, очень затратная по времени.**

**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**.

**С помощью индекса сервер базы данных использует дополнительную структуру данных (как своего рода дополнительную таблицу) для хранения в точном порядке значений соответствующего столбца и намного быстрее достигает строки, соответствующей искомому значению.**

**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.

**Индекс может быть добавлен как к столбцу, так и к 2 или более столбцам одновременно («индексы столбцов»** "***column indexes***" **и «индексы нескольких столбцов»** "***multiple column indexes***"**).**

**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/Простой индекс** - 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** / Уникальный индекс - 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"***.

***единственный индекс в этом столбце не допускает существования записей с одинаковым значением для столбца « email /электронная почта».***

***Исключением из этого правила является случай, когда столбец не имеет атрибута NOT NULL (поэтому по умолчанию он имеет значение NULL), а значение NULL может появляться в этом столбце несколько раз.***

**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.

* ***Первичный ключ*** - ***индекс, который определяет как уникальность значений этого столбца, так и наличие значения (значение NULL не принимается). Оптимально, любая таблица имеет столбец с индексом первичного ключа, и этот столбец является целым числом и имеет дополнительный атрибут AUTO\_INCREMENT.***
* **Cheie primara** /***Первичный ключ*** - 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***

1. realizeazăm tabelul ***utilizatori***, doar cu ***indexul cheie primara***
2. 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 ADDUNIQUE **INDEX** (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!!!**

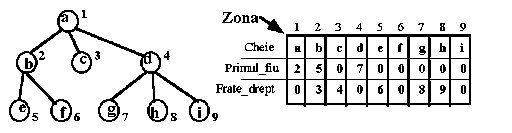
Să admitem că urmează să executăm următoarea interogare (*firstname!*).

**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. ***Будет создана «обобщенная двоичная древовидная карта», которая упростит поиск записей в списке записей в таблице.***



**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.***

**Двоичное дерево - это индекс сбалансированного дерева, это индекс, сгруппированный по листьям сбалансированного дерева. Он используется для больших индексов, по сути, это индекс индексов.**

**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.***

**На пальцах предположим, что индексы со значением от 1 до 10 хранятся в первой ветви, от 11 до 20 - во второй и т. д., Когда приходит запрос на индекс 35, используется 3-я ветвь, где находится пятый элемент.**

**Pentru un tabel mic, cu numarul de înregistrări <1000, 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!

**Для небольшой таблицы, с количеством записей <1000, преимущество не будет очевидным! Все, что вам нужно сделать, это попытаться присоединиться, добавив более (3-4 JOIN) в таблицы неиндексированных полей, и эта комбинация в какой-то момент сможет «убить» сервер!**

|  |
| --- |
| **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

***- представляет результат некоторой ключевой функции***

***- не видит расстояние по отношению к ближайшим элементам***

***- не может использоваться для операций с диапазоном «>» и «<».***

***- сравнивает весь ключ***

***- линейный при индексации***

**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))*.***

***- указатель индексов***

***- сбалансированное дерево, листья которого гарантированно имеют одинаковое количество предков***

***- поиск по интервалам, определяемыми операторами =,>,> =, <, <= или BETWEEN***

***- можно использовать для поиска ключевой префикс***

***- может использоваться в сравнениях LIKE, если аргумент LIKE является постоянной строкой, которая не начинается с символа шаблона {% (\*) или \_ (?)}***

***- Благодаря такой структуре сложность поиска по BTREE индексу соответствует уровню 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.

***Удаление / уничтожение первичных ключей (индексов PRIMARY KEY) может быть выполнено только с помощью оператора 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.

***Если такой индекс PRIMARY KEY не был создан, а таблица имеет один или несколько UNIQUE индексов, будет удален первый из них.***

**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.

***Удаление / уничтожение столбцов в таблице оказывает влияние на индексы. Таким образом, при удалении столбца из таблицы этот столбец удаляется из индекса. Удаляя все проиндексированные столбцы из таблицы, мы удаляем весь индекс.***

**DROP INDEX index\_name ON table\_name**

**DROP INDEX oras\_index on CetOraseRM**

**DEZAVANTAJE ALE INECSILOR *НЕДОСТАТКИ ИНДЕКСОВ.***

***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.***

***Во-первых, он занимает место на диске. И если это небольшая проблема для небольших баз данных, то для больших баз данных индекс может занимать больше памяти, чем вся база данных.***

***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***.

***Во-вторых, надо быть осторожным. Не стоит создавать индексы для всех полей, которые мы хотим и можем. Мы должны быть избирательными в своих действиях. И если индексы действительно ускоряют получение данных, тогда это хорошо, но нам нужно знать, что индексы замедляют скорость вычислений других операций, таких как - обновление UPDATE, вставка INSERT и удаление данных DELETE***

**Prin urmare,** **dacă aceste operațiuni sunt foarte des efectuate în baza de date, atunci utilizarea indecsilor ar trebui să fie una limitată**.

***Следовательно, если эти операции очень часто выполняются в базе данных, то следует ограничить использование индексов.***

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”.**

***Индексы FULLTEXT отличаются от всех вышеперечисленных, и их поведение существенно отличается от СУБД к СУБД. Индексы FULLTEXT полезны только для полнотекстового поиска с опцией MATCH () / AGAINST (), в отличие от трех приведенных выше (KEY или INDEX, Unique, Primary), которые обычно реализуются внутренне с использованием BTREE-деревьев (которые позволяют выбирать, сортировка или диапазоны от крайнего левого столбца) или HASH -таблиц (которые позволяют выбирать из крайнего левого столбца).***

***Если другие типы индексов являются универсальными, то индекс FULLTEXT является специализированным, поскольку он служит ограниченной цели: он используется только для функции «полнотекстового поиска».***

<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 **1 000 000** записей. У каждой записи есть уникальный идентификатор **ID**. И, допустим, нам надо вытащить записть с **ID = 530 124**. 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.



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



1. 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!!!