

MySQL

• **Partea VI. Funcționalități MySQL (II)**

Ce ne așteaptă?

1. Funcțiile definite de utilizator
2. Funcțiile încorporate
3. Declanșatorii
4. Cursorarele
5. Tranzacțiile

1. Funcții definite de utilizator

Noțiunea de funcție

- **Funcția** – o secvență denumită de cod, ce poate accepta și parametri de intrare, care returnează o valoare
- Funcțiile sunt de două tipuri:
 - Funcții încorporate – funcțiile standard stocate pe serverul MySQL
 - Funcții definite de utilizator – funcțiile create de utilizator în dependență de necesități
- Deosebirile dintre funcțiile definite de utilizator și procedurile stocate:
 - Funcțiile returnează doar o singură valoare
 - Funcțiile pot avea doar parametri de intrare
 - Funcțiile se pot folosi în interogările SQL
 - Funcțiile pot fi apelate în interiorul procedurilor însă în funcție nu se pot apela procedurile

Crearea funcțiilor

- **Sintaxa de creare a funcțiilor:**

```
DELIMITER //  
CREATE FUNCTION nume_functie (param_1 tip_param_1, param_2 tip_param_2)  
RETURNS tip_val_returnata  
DETERMINISTIC/NOT DETERMINISTIC  
BEGIN  
  DECLARE nume_variabila tip_variabila;  
  sectiune_executabila;  
  RETURN variabila_returnata;  
END//
```

- **O funcție poate să nu conțină parametri sau să poate conțină unu sau mai mulți**
- **RETURNS precizează tipul valorii returnate de către funcție**
- **DETERMINISTIC precizează că valoarea returnata va fi aceeași pentru aceleași date de intrare, iar NOT DETERMINISTIC (implicit) că aceasta poate varia**
- **DECLARE permite declararea variabilelor locale**
- **RETURN returnează valoarea funcție**

Apelarea funcțiilor

- **Sintaxa de apelarea a funcțiilor**

```
SELECT nume_funcție (param_1, param_2)
```

sau

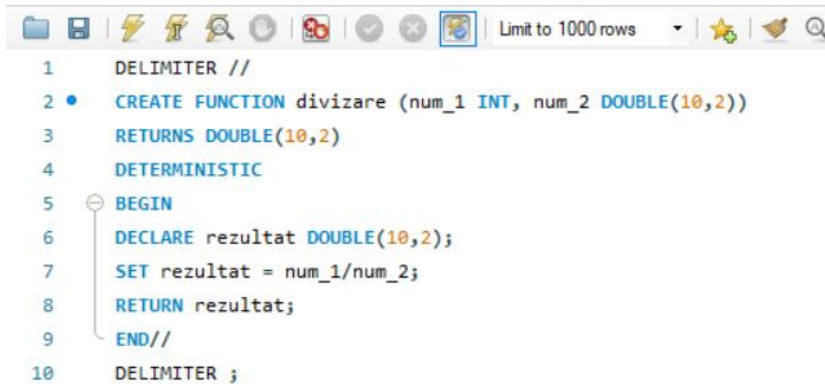
```
SET @nume_variabila =nume_funcție (param_1, param_2)
```

```
SELECT @nume_variabila
```

- **O funcție poate fi apelată și din interiorul unei proceduri stocate sau a unei alte funcții**
- **Numărul și tipul parametrilor trebuie să coincidă cu cei declarați la crearea funcției**

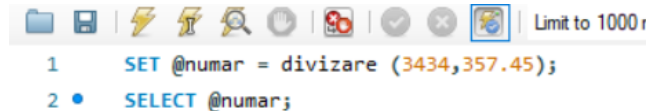
Exemplu de creare și apelare a funcției

- Crearea unei funcții de divizarea a două numere

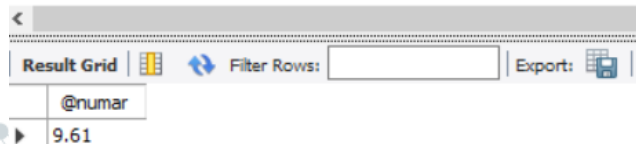


```
1 DELIMITER //
2 • CREATE FUNCTION divizare (num_1 INT, num_2 DOUBLE(10,2))
3 RETURNS DOUBLE(10,2)
4 DETERMINISTIC
5 BEGIN
6 DECLARE rezultat DOUBLE(10,2);
7 SET rezultat = num_1/num_2;
8 RETURN rezultat;
9 END//
10 DELIMITER ;
```

- Apelarea funcției de divizarea a două numere



```
1 SET @numar = divizare (3434,357.45);
2 • SELECT @numar;
```



@numar
9.61

Aplicarea funcției asupra datelor bazei de date

- Vizualizarea tuturor datelor despre toate țările din tabelul tari cu adăugarea și a densității populației pentru fiecare țară



```
1 SELECT id_tara, nume_tara, capitala, suprafata, populatie, divizare(populatie, suprafata) AS densitate FROM tari;
```

A screenshot of a MySQL query editor window showing the result grid of the previous query. The toolbar at the top includes icons for file operations, search, and execution. A dropdown menu shows 'Limit to 1000 rows'. The query text area contains the following SQL statement:

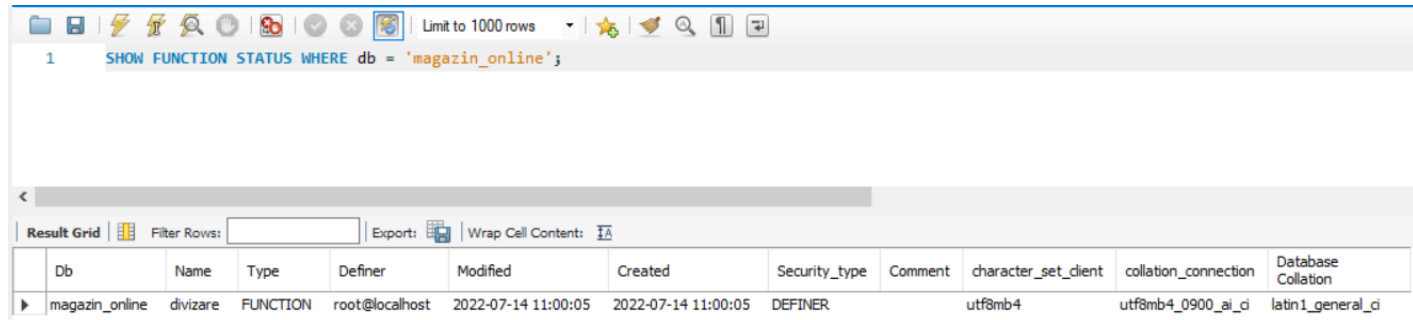
	id_tara	nume_tara	capitala	suprafata	populatie	densitate
▶	1	Italia	Roma	294140.00	60461826	205.55
	2	Romania	Bucuresti	238391.00	19483360	81.73
	3	Franta	Paris	551695.00	65480710	118.69
	4	Germania	Berlin	357114.00	82438639	230.85
	5	Moldova	Chisinau	33846.00	4029750	119.06

Vizualizarea tuturor funcțiilor

- Pentru vizualizarea tuturor funcțiilor unei baze de date se va utiliza sintaxa:

`SHOW FUNCTION STATUS WHERE db = nume_baza`

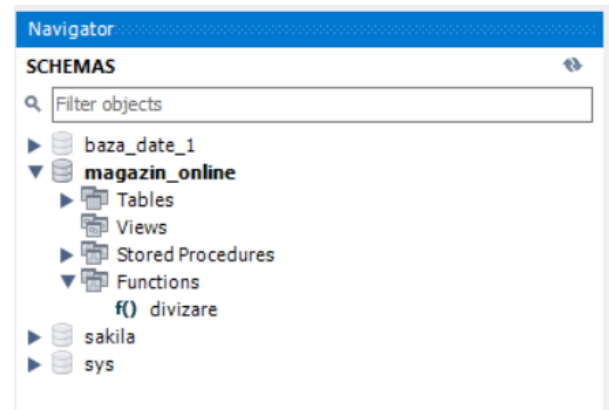
- Exemplu:



The screenshot shows the MySQL Workbench interface. The command editor contains the query: `SHOW FUNCTION STATUS WHERE db = 'magazin_online';`. Below the editor, the 'Result Grid' tab is active, displaying a table with the following data:

Db	Name	Type	Definer	Modified	Created	Security_type	Comment	character_set_client	collation_connection	Database Collation
magazin_online	divizare	FUNCTION	root@localhost	2022-07-14 11:00:05	2022-07-14 11:00:05	DEFINER		utf8mb4	utf8mb4_0900_ai_ci	latin1_general_ci

- În MySQL Workbench funcțiile ca și alte obiecte pot fi vizualizate în câmpul de navigare

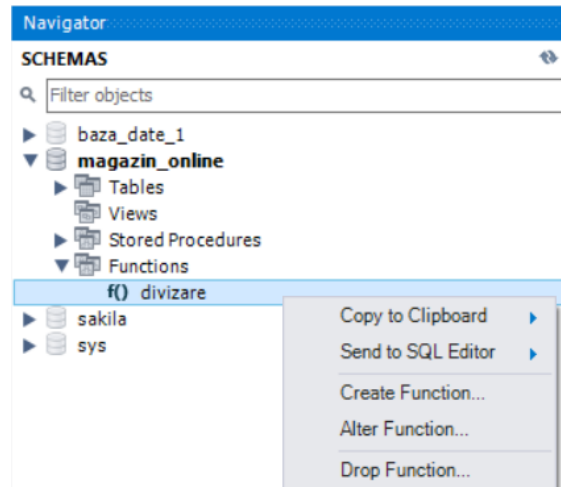


Ștergerea unei funcții

- Pentru ștergerea unei funcții se utilizează sintaxa:

`DROP FUNCTION IF EXISTS nume_funcție`

- În MySQL Workbench o funcție se poate șterge prin selectarea opțiunii Drop Function... când se dă clic dreapta pe numele funcției în câmpul de navigare



2. Funcții încorporate

Clasificarea funcțiilor încorporate

- **Funcția încorporată (built-in)** – funcție standard stocată pe serverul MySQL
- **Funcțiile încorporate pot fi**
 - **Funcțiile scalare** – acceptă o valoare ca parametru de intrare și returnează o valoare ca ieșire
 - **Funcțiile agregate** – manipulează mai multe valori dar și returnează o singură valoare ca ieșire
- **Funcțiile încorporate se mai împart în 5 categorii:**
 - **Funcțiile de manipulare a timpului și a datei**
 - **Funcțiile de manipulare a stringurilor**
 - **Funcțiile de manipulare a valorilor numerice**
 - **Funcțiile de sumare**
 - **Funcțiile de control**

Modificarea setărilor tabelului

- **now()** – permite obținerea datei și orei curente

```
SELECT now()
```

- **curtime()** – permite obținerea orei curente

```
SELECT curtime()
```

- **curdate()** – permite obținerea datei curente

```
SELECT curdate()
```

- **date_add()** – permite adăugarea unei perioade la timpul curent

```
SELECT date_add(now(), INTERVAL valoare_timp unitate_timp)
```

- **date_sub()** – permite scoaterea unei perioade de la timpul curent

```
SELECT date_sub(now(), INTERVAL valoare_timp unitate_timp)
```

valoare_timp reprezintă valoarea numerică iar **unitate_timp** – unitatea de măsurare a timpului ex: `INTERVAL 10 day` sau `INTERVAL 15 hour`

- **date_format()** – permite obținerea formatului timpului conform necesităților
`SELECT date_format(val_timp, șablon)`
- **val_timp** date în format datetime inclusiv poate fi `now()`, `curtime()`, `curdate()`, etc
- **șablon** formatul dorit al timpului cu utilizarea unui string cu diferite notații ex:
`SELECT date_format(now, '%d-%M-%Y %H-%i-%s')`
- **Semnificația simbolurilor din șablon:**

Semn	Semnificație	Semn	Semnificație
%d	Ziua în format numeric, ex: 01	%W	Ziua săptămânii complet, ex: Thursday
%D	Ziua în format numeric cu sufix, ex: 14th	%a	Ziua săptămânii prescurtat, ex: Thu
%m	Luna în format numeric, ex: 08	%H	Ora în format 24 ore, ex: 15
%M	Luna în format text, ex: August	%h	Ora în format 12 ore, ex: 03
%b	Luna în format text prescurtat, ex: Aug	%p	Semnul AM sau PM
%y	Anul în format numeric cu 2 cifre, ex: 21	%i	Minute, ex: 45
%Y	Anul în format numeric cu 4 cifre, ex: 2021	%s	Secunde, ex: 37

Funcții de manipulare a timpului și datei (3)

- Funcții de extragere a unei părți a datei sau timpului, exemplu extragerea orei

```
SELECT HOUR(now());
```

- Lista funcțiilor de extragere:

Sintaxa	Descriere	Rezultat
DAYOFMONTH(dt)	ziua prezentată în forma numerică	25
DAYNAME(dt)	numele zilei	Thursday
MONTH(dt)	luna în forma numerică	9
MONTHNAME(dt)	luna în forma textuală	September
YEAR(dt)	anul în forma numerică din patru cifre	2014
HOUR(dt)	ceas în format de 24 de ore	15
MINUTE(dt)	minute	36
SECOND(dt)	secunde	3
DAYOFYEAR(dt)	numărul de ordine al zilei în an	268

Funcții de manipulare a stringurilor (1)

- **concat(str1,str2)** – permite concatenarea a 2 stringuri

```
SELECT now('salut ', 'tuturor');
```

- **replace(text_initial, partea_veche, partea noua)** – permite substituirea unei părți a stringului cu alta

```
SELECT replace('salut tuturor', 'salut', 'buna');
```

- **insert(text_initial, poziție, lungime, text_inclus)** – permite includerea unei porțiuni de text într-un text existent pe o anumită poziție

```
SELECT insert('salut tuturor', 7, 0, 'calduros ');
```

- **left(string, numar)/right(string, numar)** – permite selectarea unui anumit număr de caractere de începând de la stânga/dreapta stringului

```
SELECT left('salut tuturor', 5);
```

Funcții de manipulare a stringurilor (2)

- **mid(string, pozitie_start, numar)** – permite selectarea unui anumit număr de caractere de începând cu poziția specificată

```
SELECT mid('salut tuturor', 7, 3);
```

- **substring(string, pozitie_start)** – permite selectarea stringului începând cu poziția specificată până la sfârșit

```
SELECT substring('salut tuturor', 7);
```

- **trim(string)/ltrim(string)/rtrim(string)** – permite excluderea spațiilor goale din ambele/stânga/dreapta părți/parte a stringului

```
SELECT trim('  salut tuturor ');
```

```
SELECT ltrim('  salut tuturor ');
```

```
SELECT rtrim('  salut tuturor ');
```

- **length(string)** – permite determinarea numărului de caractere în string

```
SELECT length('salut tuturor');
```

Funcții de manipulare a valorilor numerice (1)

- Funcții ce permite prelucrarea datelor numerice

Funcție	Descriere	Exemplu	Rezultat
ABS()	returnează valoarea absolută a numărului introdus	select ABS(-434)	434
ACOS()	returnează arccos cosinus al numărului introdus	select ACOS(1/2)	1.04719755
ASIN()	returnează arcsinus al numărului introdus	select ASIN(1/2)	0.52359877
ATAN()	returnează arctangentă al numărului introdus	select ATAN(1/2)	0.46364760
CEIL()	rotunjește valoarea zecimală la următorul număr întreg	select CEIL(5.7)	6
CEILING()	în MySQL CEILING și CEIL sunt sinonime	select CEILING(5.7)	6
CONV()	conversia dintr-o notație numerică într-alta	select CONV(10, 2, 10)	2
COS()	returnează cosinus-ul numărului introdus	select COS(1/2)	0.8775825
COT()	returnează cotangenta numărului introdus	select COT(1/2)	1.8304877

Funcții de manipulare a valorilor numerice (2)

Funcție	Descriere	Exemplu	Rezultat
CRC32()	calculează valoarea <i>cyclic redundancy check</i> (verificare redundanță ciclică)	select CRC32(10)	2707236321
DEGREES()	convertește radiani în grade	select DEGREES(1/2)	28.64788975654116
EXP()	returnează exponentul numărului introdus	SELECT EXP(2)	7.389056098
FLOOR()	rotunjește valoarea zecimală la numărul întreg precedent	select FLOOR(1.7)	1
LN()	returnează logaritmul numărului introdus	select LN(10)	2.302585092994046
LOG10()	returnează logaritmul numărului introdus cu baza 10	select LOG10(100)	2
LOG2()	returnează logaritmul numărului introdus cu baza 2	select LOG2(65536)	16
LOG()	returnează logaritmul natural al primei cifre; cu doi parametri, returnează logaritmul primului pentru baza celuiilalt	select LOG(10,100)	2
MOD()	returnează restul împărțirii	select MOD(234, 10);	4
OCT()	returnează numărul zecimal în reprezentarea octală	select OCT(10);	12

Funcții de manipulare a valorilor numerice (3)

Funcție	Descriere	Exemplu	Rezultat
PI()	returnează valoarea PI	select PI();	3.141593
POW()	crește prima valoare introdusă la a doua valoare introdusă ca exponent	select POW(2, 2);	4
POWER()	în MySQL POW și POWER sunt sinonime	select POWER(2, 2);	4
RADIANS()	convertește valoarea în radiani	select RADIANS(90);	1.5707963267948966
RAND()	returnează valoarea obținută ocazional între 0 și 1	select RAND();	0.576998468094504
ROUND()	rotunjește valoarea zecimală la număr întreg	select ROUND(23.23);	23
SIGN()	returnează semnul valorii; 1 pentru pozitiv, -1 pentru negativ	select SIGN(-25);	-1
SIN()	returnează sinus pentru valoarea introdusă	select SIN(90);	0.8939966636005579
SQRT()	returnează rădăcina pătrată a valorii introduse	select SQRT(25);	5
TAN()	returnează tangenta numărului introdus	select TAN(90);	-1.995200412208242
TRUNCATE()	elimină un anumit număr al numărului zecimal	select TRUNCATE(23.3432423, 2);	23.34

Funcții de sumare

- **Funcțiile de sumare includ funcțiile de agregare ce procesează mai multe date incluse într-un sigur parametru de intrarea - denumirea unei coloane a tabelului**
- **count(ume_col) – determină numărul de valori în coloana trecută ca parametru**

```
SELECT count(ume_coloana) FROM ume_tabel;
```
- **sum(ume_col) – determină suma tuturor valorilor unei coloane cu date numerice**

```
SELECT sum(ume_coloana) FROM ume_tabel;
```
- **avg(ume_col) – determină valoarea medie a unei coloane cu date numerice**

```
SELECT avg(ume_coloana) FROM ume_tabel;
```
- **max(ume_col) – determină valoarea maximă a unei coloane cu date numerice**

```
SELECT max(ume_coloana) FROM ume_tabel;
```
- **min(ume_col) – determină valoarea minimă a unei coloane cu date numerice**

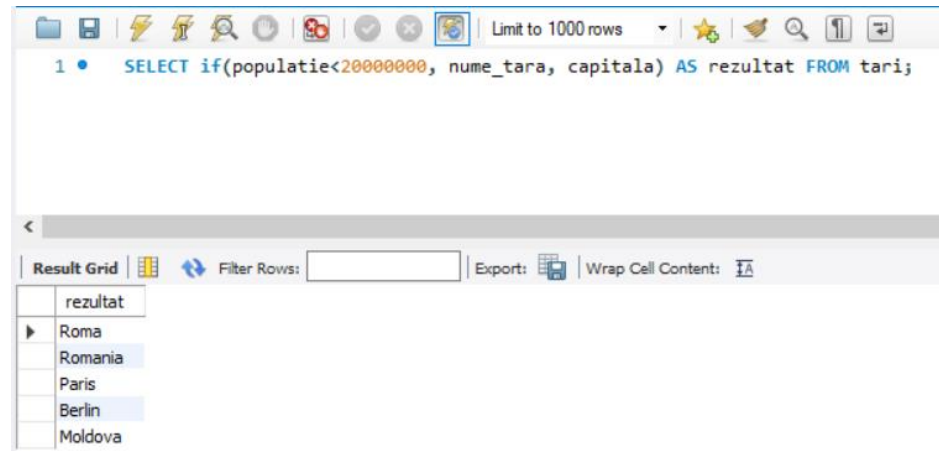
```
SELECT min(ume_coloana) FROM ume_tabel;
```

Funcția de control if()

- Funcțiile de control permit controlul modului în care vor fi tratate elementele unei interogări
- if() - permite în dependență de o condiție selectarea unuia din două elemente ce va fi transmit interogării

`if(conditia, elem_adevarat, elem_fals)`

- Dacă `conditia` se îndeplinește atunci se selectează elementul `elem_adevarat`, iar dacă nu `elem_fals`
- Exemplu: Dacă în tabelul `tari`, populația este mai mica de 2000000, atunci se afișează numele țării dacă nu - numele capitalei

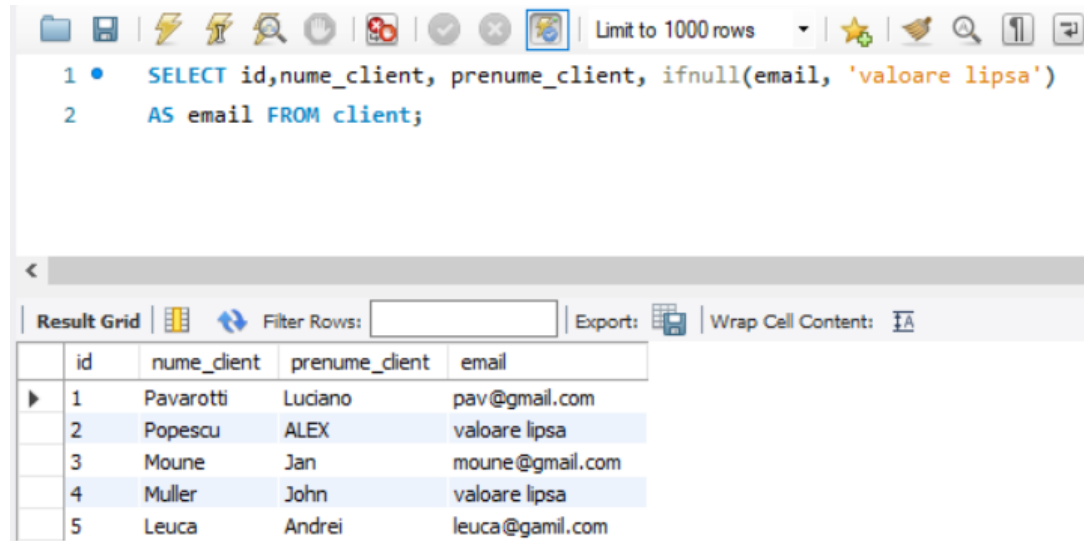


Funcția de control ifnull()

- **ifnull()** - verifica daca primul parametru este nu atunci returnează al doilea parametru, iar daca nu este nual atunci returnează acest prim parametru.

`ifnull(param_1, param_2)`

- **Exemplu:** preventiv se adaugă o coloana email ce accepta valori nule în tabelul client



The screenshot shows a MySQL query editor window. The query is as follows:

```
1 • SELECT id, nume_client, prenume_client, ifnull(email, 'valoare lipsa')
2 AS email FROM client;
```

Below the query, the results are displayed in a table with 5 rows. The columns are id, nume_client, prenume_client, and email. The email column contains the original email values or 'valoare lipsa' for null values.

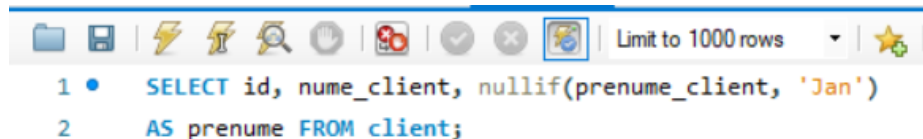
	id	nume_client	prenume_client	email
▶	1	Pavarotti	Luciano	pav@gmail.com
	2	Popescu	ALEX	valoare lipsa
	3	Moune	Jan	moune@gmail.com
	4	Muller	John	valoare lipsa
	5	Leuca	Andrei	leuca@gamil.com

Funcția de control nullif()

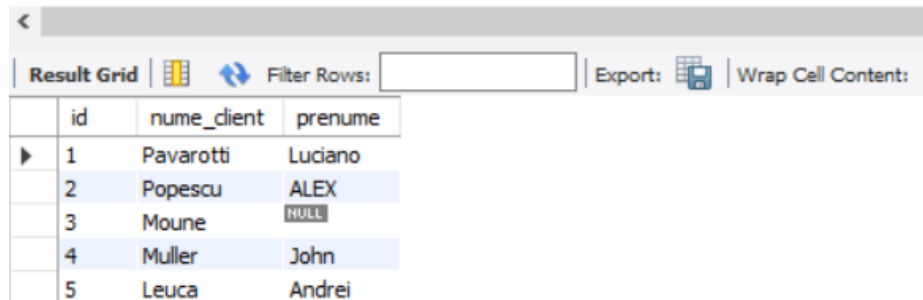
- **nullif()** - verifica dacă parametrii sunt identici atunci returnează null iar dacă diferiți atunci returnează primul parametru

`nullif(param_1, param_2)`

- **Exemplu:**



```
1 • SELECT id, nume_client, nullif(prenume_client, 'Jan')
2   AS prenume FROM client;
```



	id	nume_client	prenume
▶	1	Pavarotti	Luciano
	2	Popescu	ALEX
	3	Moune	NULL
	4	Muller	John
	5	Leuca	Andrei

3. Declanșatorii

Noțiune de declanșator

- **Declanșatorul (trigger)** – o secvență de cod atașată unui tabel ce se execută automat când are loc un eveniment asupra tabelului
- Evenimentul reprezintă o acțiune de înscriere, de actualizare sau de ștergere a datelor
- În funcție de tipul operație pentru care se execută declanșatorii sunt:
 - INSERT – declanșatorul se execută când are loc înscrierea datelor noi
 - UPDATE – declanșatorul se execută când are loc actualizarea datelor
 - DELETE – declanșatorul se execută când are loc ștergerea datelor
- În funcție de momentul acțiunii declanșatorii sunt:
 - BEFORE – se declanșează înainte efectuarea propriu zisă a operației
 - AFTER – se declanșează după efectuarea propriu zisă a operației
- Declanșatorii pot apela datele implicate în operații prin adăugarea prefixelor:
 - OLD – pentru datele anterioare operației (declanșatorii UPDATE și DELETE)
 - NEW – pentru datele posterioare operației (declanșatorii UPDATE și INSERT)

Crearea declanșatorilor

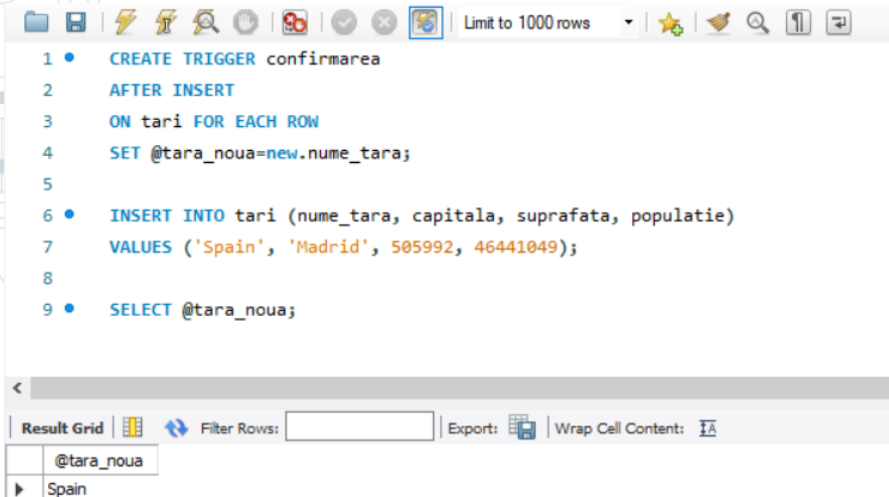
- **Sintaxa de crearea a declanșatorilor:**

```
CREATE TRIGGER nume_declanșator timp_declanșator actiune_declansator  
ON nume_tabel FOR EACH ROW  
ordine_declanșator nume_declașator_existent  
BEGIN  
corp_declansator  
END
```

- **timp_declanșator** poate avea valorile **AFTER** sau **BEFORE**
- **actiune_declanșator** poate avea valorile **INSERT**, **UPDATE** sau **DELETE**
- **ordine_declanșator** se utilizează în cazul în care există deja un declanșator pe acțiunea respectivă și poate avea valorile **FOLLOWS** sau **PRECEDES**
- **corp_declanșator** include interogările ce se vor executa când se apelează declanșatorul
- **BEGIN** și **END** se utilizează în cazul corpul declanșatorului constă din mai multe interogări

Exemplu declanșator

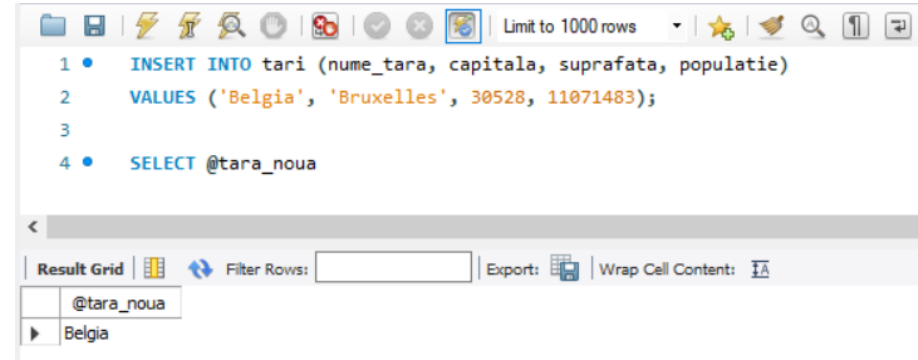
- Crearea unui declanșator INSERT ce va atribui numele tari noi incluse unei variabile
- După interogarea de înscrierea se va selecta variabila pentru a se confirma înscrierea



```
1 • CREATE TRIGGER confirmarea
2   AFTER INSERT
3   ON tari FOR EACH ROW
4   SET @tara_noua=new.ume_tara;
5
6 • INSERT INTO tari (ume_tara, capitala, suprafata, populatie)
7   VALUES ('Spain', 'Madrid', 505992, 46441049);
8
9 • SELECT @tara_noua;
```

Result Grid

@tara_noua
Spain



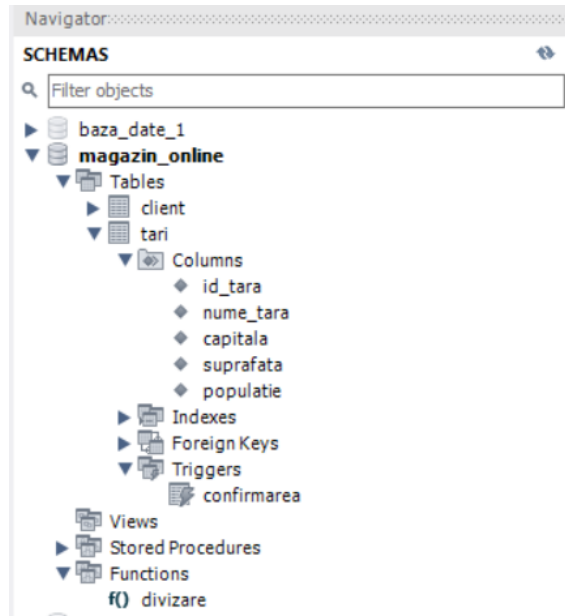
```
1 • INSERT INTO tari (ume_tara, capitala, suprafata, populatie)
2   VALUES ('Belgia', 'Bruxelles', 30528, 11071483);
3
4 • SELECT @tara_noua
```

Result Grid

@tara_noua
Belgia

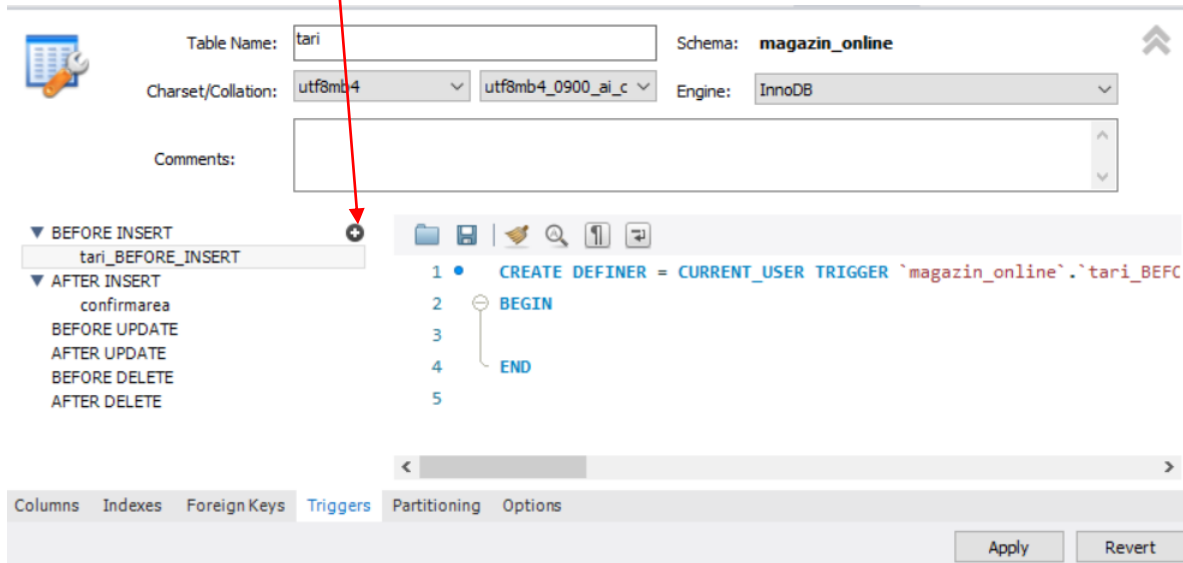
Vizualizarea declanșatorilor

- Pentru vizualizarea tuturor declanșatorilor unei baze de date se va scrie sintaxa
`SHOW TRIGGERS FROM nume_baza;`
- În MySQL Workbench declanșatorii pot fi vizualizați în opțiunea Triggers a tabelelor din câmpul de navigare



Crearea declanșatorilor în Workbench

- Se inițiază procedura de modificare a tabelului (Alter Table...)
- În fereastra care apare se selectează tab-ul Triggers
- Se acționează simbolul + de pe linia tipului de declanșator dorit
- În fereastra alăturată se include codul corpului declanșatorului

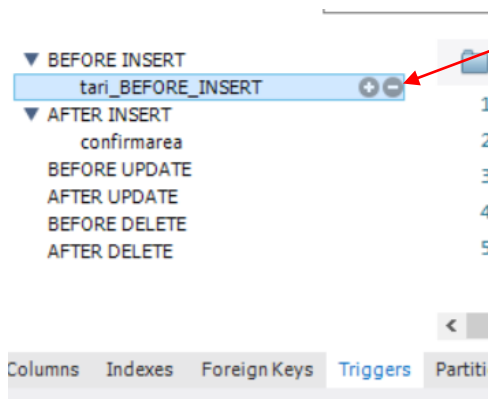


Ștergerea declanșatorului

- Un declanșator se șterge cu sintaxa:

`DROP TRIGGER IF EXISTS nume_declanșator`

- În Workbench un declanșator se șterge prin acționarea semnului – din linia declanșatorului



4. Cursurile

Noțiunea de cursor

- **Cursorul este un mecanism de citire a datelor mai multor linii și atribuirea acestora unei variabile sub forma unei singure linii**
- **Cursorul se utilizează în corpul procedurilor stocate, funcțiilor definite de utilizator și a declanșatoarelor**
- **Cursorul permite doar citirea datelor nu și modificarea acestora**
- **Cursorul se utilizează cu interogarea SELECT**
- **Etapele utilizării cursorului**
 - **Declararea variabilelor ce vor fi utilizate cu cursorul**
 - **Declararea cursorului**
 - **Declararea handler- ului de finisare**
 - **Deschiderea cursorului**
 - **Extragerea informației**
 - **Închiderea cursorului**

Declararea variabilelor cursorului și handler-ului

- Declararea a 2 variabile, una pentru stocarea datelor cu valoarea implicită un string gol și a doua pentru determinarea ultimei linii de date cu valoarea implicită 0

```
DECLARE date VARCHAR(100) DEFAULT '';  
DECLARE finis INT DEFAULT 0;
```

- Declararea cursorului

```
DECLARE nume_cursor CURSOR FOR interogare_SELECT;
```

- Cursorul va citi datele linie după linie și când va ajunge la ultima linie va genera o eroare deoarece nu există următoarea linie
- Pentru evitarea acestei erori se creează un handler ce modifică variabila `finis` când se ajunge la ultima linie

```
DECLARE CONTINUE HANDLER NOT FOUND SET finis= 1;
```

Deschiderea cursorului, extragerea datelor și închiderea lui

- Pentru utilizarea cursorul necesită a fi deschis

`OPEN nume_cursor`

- Sintaxa generală de extragere a datelor

`FETCH nume_cursor INTO date`

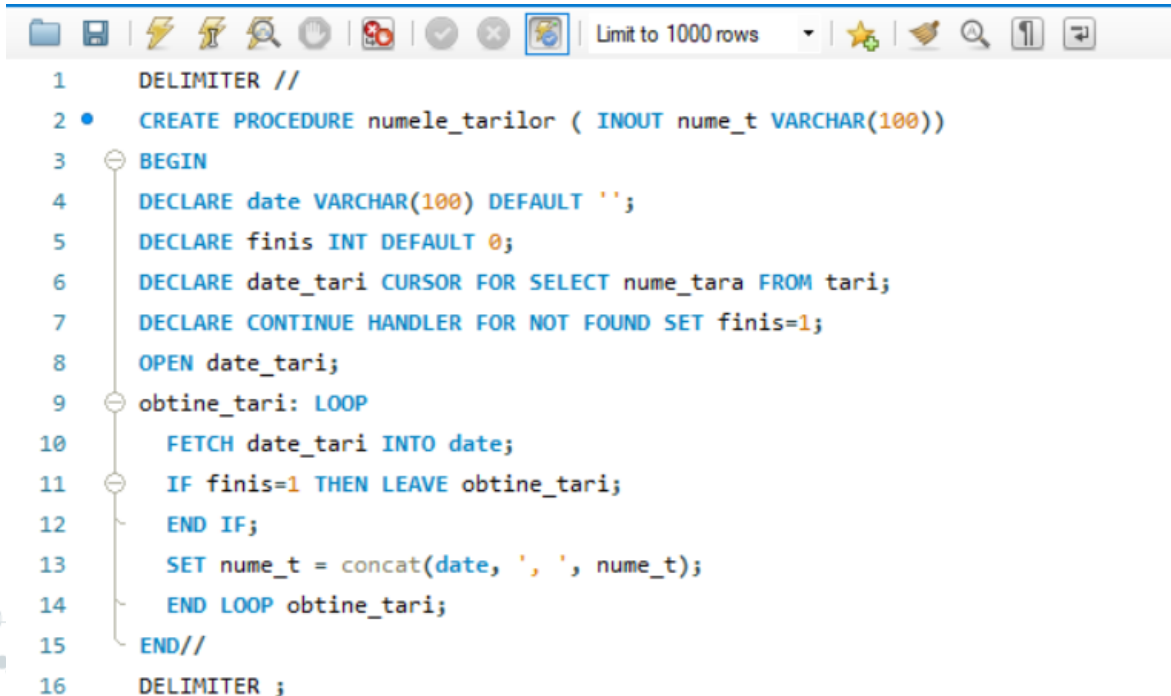
- `date` este variabila de stocare declarata la început
- Deoarece cursorul permite citirea mai multor linii, sintaxa de extragere a datelor se include într-o buclă în care se mai include verificarea variabilei de detectare a ultimei linii și procedura de concatenarea a datelor variabilei de stocare
- Pentru eliberarea memorie cursorul trebuie închis

`CLOSE nume_cursor`

- Odată închis cursorul poate fi redeschis fără a mai fi necesară declararea sa

Exemplu de utilizarea a cursorului (1)

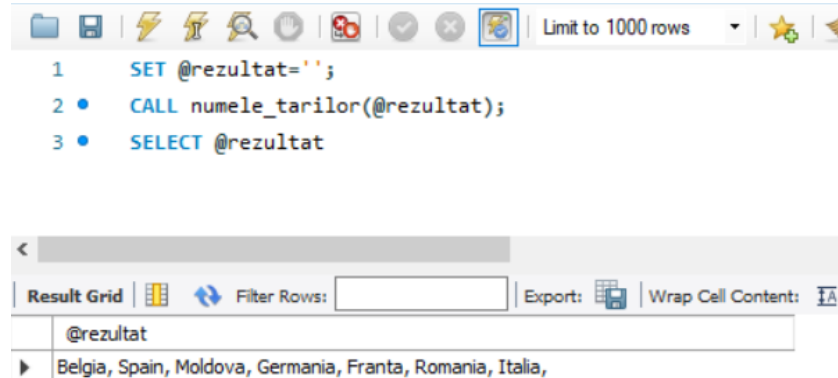
- Crearea unei proceduri ce va citi numele tuturor țărilor din tabelul tari și le va atribui unei variabile sub forma unui string cu separare prin virgulă



```
1  DELIMITER //
2  • CREATE PROCEDURE numele_tarilor ( INOUT nume_t VARCHAR(100))
3  BEGIN
4      DECLARE date VARCHAR(100) DEFAULT '';
5      DECLARE finis INT DEFAULT 0;
6      DECLARE date_tari CURSOR FOR SELECT nume_tara FROM tari;
7      DECLARE CONTINUE HANDLER FOR NOT FOUND SET finis=1;
8      OPEN date_tari;
9      obtine_tari: LOOP
10         FETCH date_tari INTO date;
11         IF finis=1 THEN LEAVE obtine_tari;
12     END IF;
13     SET nume_t = concat(date, ', ', nume_t);
14     END LOOP obtine_tari;
15 END//
16 DELIMITER ;
```


Exemplu de utilizarea a cursorului (2)

- Declararea unei variabile, apelarea procedurii cu variabila declarată și afișarea rezultatului



The screenshot displays a MySQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL editor contains the following code:

```
1 SET @rezultat='';  
2 CALL numele_tarilor(@rezultat);  
3 SELECT @rezultat
```

Below the editor, the 'Result Grid' tab is active, showing the output of the query:

@rezultat
Belgia, Spain, Moldova, Germania, Franta, Romania, Italia,

5. Tranzacțiile

Noțiune de tranzacție

- Tranzacțiile sunt mecanisme de gruparea a mai multor comenzi SQL într-o singură operație pentru a asigura integritatea datelor
- Tranzacțiile se utilizează cu scopul de a:
 - asigura efectuarea tuturor comenzilor asupra datelor sau respingerea tuturor când una din comenzi eșuează (“totul sau nimic”)
 - interzice modificarea aceleași date de către doi utilizatori diferiți în același moment de timp
- Tranzacțiile sunt suportate doar în tabelele InnoDB
- O interogarea SQL este implicit în mod autocommit adică este o tranzacție mică care fie se va executa, fie nu se va executa, dar nu se poate spune că se va executa pe jumătate
- Gruparea mai multor interogări SQL într-o tranzacție se poate realiza prin:
 - Dezactivarea modului autocomit
 - Utilizarea expresiei **START TRANSACTION**

Controlul modului autocommit

- Pentru a dezactiva modul autocommit, se setează variabila de sistem autocommit la valoarea 0 sau OFF
- Pentru aprobarea execuției interogărilor în lipsa modului autocommit se utilizează cuvântul-cheie COMMIT
- Pentru anularea execuției interogărilor în lipsa modului autocommit se utilizează cuvântul-cheie ROLLBACK
- Sintaxa tranzacției autocommit:

```
SET autocommit=0;  
interogări de manipulare;  
COMMIT/ROLLBACK;
```
- Neajunsul: Imediat după COMMIT/ROLLBACK începe o nouă tranzacție deci orice acțiune asupra datelor bazei va trebui confirmată sau anulată și nu doar un grup de interogări cum ne-am fi dorit

Exemplu de setarea autocommit

- Exemplu cu anularea interogărilor

Limit to 1000 rows

```

1 • set autocommit = 0;
2 • UPDATE client SET prenume_client='Alex' WHERE prenume_client='ALEX';
3 • INSERT INTO client (nume_client, prenume_client, tara) VALUES ('Popovici', 'Vlad', 2);
4 • ROLLBACK;
5 • SELECT * FROM client;
  
```

Result Grid

	id	nume_client	prenume_client	tara	email
▶	1	Pavarotti	Luciano	1	pav@gmail.com
	2	Popescu	ALEX	2	NULL
	3	Moune	Jan	3	moune@gmail.com
	4	Muller	John	4	NULL
	5	Leuca	Andrei	5	leuca@gamil.com
•	NULL	NULL	NULL	NULL	NULL

- Exemplu cu confirmarea interogărilor

Limit to 1000 rows

```

1 • set autocommit = 0;
2 • UPDATE client SET prenume_client='Alex' WHERE prenume_client='ALEX';
3 • INSERT INTO client (nume_client, prenume_client, tara) VALUES ('Popovici', 'Vlad', 2);
4 • COMMIT;
5 • SELECT * FROM client;
  
```

Result Grid

	id	nume_client	prenume_client	tara	email
▶	1	Pavarotti	Luciano	1	pav@gmail.com
	2	Popescu	Alex	2	NULL
	3	Moune	Jan	3	moune@gmail.com
	4	Muller	John	4	NULL
	5	Leuca	Andrei	5	leuca@gamil.com
	12	Popovici	Vlad	2	NULL
•	NULL	NULL	NULL	NULL	NULL

Tranzacția unică

- O tranzacție unică începe cu comanda **START TRANSACTION** și utilizează comenzile **COMMIT** sau **ROLLBACK** pentru confirmarea/anularea interogărilor
- Sintaxa creării unei tranzacții unice

```
START TRANSACTION;  
interogări de manipulare;  
COMMIT/ROLLBACK;
```

- Interogările ce urmează după comenzile **COMMIT/ROLLBACK** se vor afla în mod autocommit deci vor acționa ca o mini tranzacție ce nu necesită a fi confirmată sau anulată

Exemplu de crearea a tranzacțiilor

- Crearea tranzacției cu anulare

```
1 • START TRANSACTION;
2 • UPDATE client SET prenume_client='Alexandru' WHERE prenume_client='Alex';
3 • INSERT INTO client (nume_client, prenume_client, tara) VALUES ('Close', 'Sebastian', 4);
4 • ROLLBACK;
5 • SELECT * FROM client;
```

Result Grid

	id	nume_client	prenume_client	tara	email
▶	1	Pavarotti	Luciano	1	pav@gmail.com
	2	Popescu	Alex	2	NULL
	3	Moune	Jan	3	moune@gmail.com
	4	Muller	John	4	NULL
	5	Leuca	Andrei	5	leuca@gamil.com
	12	Popovici	Vlad	2	NULL
*	NULL	NULL	NULL	NULL	NULL

- Crearea tranzacției cu confirmare

```
1 • START TRANSACTION;
2 • UPDATE client SET prenume_client='Alexandru' WHERE prenume_client='Alex';
3 • INSERT INTO client (nume_client, prenume_client, tara) VALUES ('Close', 'Sebastian', 4);
4 • COMMIT;
5 • SELECT * FROM client;
```

Result Grid

	id	nume_client	prenume_client	tara	email
▶	1	Pavarotti	Luciano	1	pav@gmail.com
	2	Popescu	Alexandru	2	NULL
	3	Moune	Jan	3	moune@gmail.com
	4	Muller	John	4	NULL
	5	Leuca	Andrei	5	leuca@gamil.com
	12	Popovici	Vlad	2	NULL
	14	Close	Sebastian	4	NULL
*	NULL	NULL	NULL	NULL	NULL

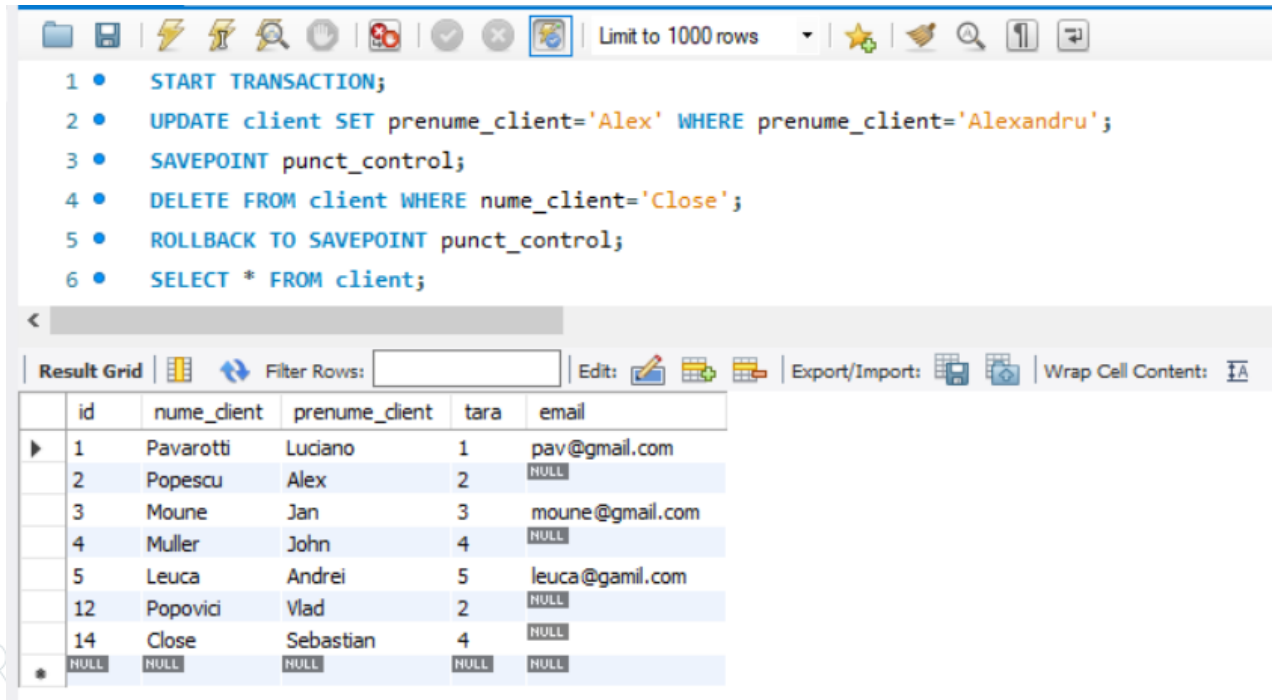
Puncte de control

- Punctele de control reprezintă niște puncte intermediare în interiorul tranzacție ce permit confirmarea sau anularea doar a unei porțiuni a tranzacției
- Pentru crearea punctelor de control se utilizează cuvântul cheie **SAVEPOINT**
- **Sintaxa tranzacție cu puncte de control**

```
START TRANSACTION;  
interogări de manipulare 1;  
SAVEPOINT nume_punct;  
interogări de manipulare 2;  
ROLLBACK TO SAVEPOINT nume_punct;
```

- În cazul punctelor de control se utilizează **ROLLBACK TO SAVEPOINT** pentru a executa interogările până la punctul de control și anularea celor ce urmează după
- Pentru confirmarea sau anularea tuturor interogărilor indiferent de punctul de control se utilizează **COMMIT/ROLLBACK**

Exemplu utilizarea a punctelor de control



The screenshot displays a MySQL client window with a script editor and a result grid. The script editor contains the following SQL commands:

```
1 • START TRANSACTION;  
2 • UPDATE client SET prenume_client='Alex' WHERE prenume_client='Alexandru';  
3 • SAVEPOINT punct_control;  
4 • DELETE FROM client WHERE nume_client='Close';  
5 • ROLLBACK TO SAVEPOINT punct_control;  
6 • SELECT * FROM client;
```

The result grid shows the output of the final SELECT statement. It includes a toolbar with options like 'Result Grid', 'Filter Rows', 'Edit', 'Export/Import', and 'Wrap Cell Content'. The data is presented in a table with the following columns: id, nume_client, prenume_client, tara, and email.

id	nume_client	prenume_client	tara	email
1	Pavarotti	Luciano	1	pav@gmail.com
2	Popescu	Alex	2	NULL
3	Moune	Jan	3	moune@gmail.com
4	Muller	John	4	NULL
5	Leuca	Andrei	5	leuca@gamil.com
12	Popovici	Vlad	2	NULL
14	Close	Sebastian	4	NULL
*	NULL	NULL	NULL	NULL

Concurența în tabele InnoDB

- **Concurența** – accesarea și procesarea datelor de către 2 clienți în același moment de timp
- În tabele InnoDB odată ce un client a inițiat o tranzacție, atunci până la finalizarea tranzacției tabelul va fi blocat în una din două forme:
 - **Blocare partajată** – ceilalți clienți pot doar citi (la nivel de linie) datele anterioare tranzacției și doar clientul care a inițiat interogarea poate modifica datele
 - **Blocare exclusivă** – ceilalți clienți nu pot nici citi și nici modifica datele (la nivel de linie) ci doar clientul care a inițiat interogarea
- **SELECT ... LOCK IN SHARE MODE** – interogarea de selectare a datelor dacă acestea sunt în interiorul altei tranzacții și se așteaptă finalizarea execuției ei
- **SELECT ... FOR UPDATE** – interogarea de selectare a datelor cu blocarea acestora în interiorul tranzacției pentru alți clienți până la finalizarea tranzacției

Exemplu de concurență în tabele InnoDB

- Exemplu de concurență în cazul a 2 tranzacții (colorate verde și orange) create de clienții A și B. (Rezultatele comenzilor SELECT sunt prezentate cu roșu)

Client A	Client B	Time
USE nume_baza INSERT INTO table1 (colA, colB) VALUES (1, 10) SELECT * FROM table1 colA = 1 colB = 10	USE nume_baza	0
START TRANSACTION UPDATE table1 SET colB=11 WHERE colA=1		1
SELECT * FROM table1 colA = 1 colB = 11	SELECT * FROM table1 colA = 1 colB = 10	2
	START TRANSACTION UPDATE table1 SET colB=colB+3 WHERE colA=1	3
COMMIT		4
SELECT * FROM table1 colA = 1 colB = 11	SELECT * FROM table1 colA = 1 colB = 14	5
	ROLLBACK	6
SELECT * FROM table1 colA = 1 colB = 11	SELECT * FROM table1 colA = 1 colB = 11	7

Concurența în tabele MyISAM

- Tabelele MyISAM nu suportă tranzacțiile deci pentru rezervarea unui tabel pentru un client se procedează la blocarea acestuia

- Sintaxa de blocare a unui tabel

```
LOCK TABLES nume_tabel optiune;
```

- Sintaxa de blocare a mai multor tabele

```
LOCK TABLES nume_tabel_1 optiune, nume_tabel_2 optiune, ...;
```

- **optiune** poate avea una din valorile:

- **READ** – alți clienți pot citi dar nu pot modifica datele
- **WRITE** – alți clienți nu pot nici citi și nici modifica datele

- Sintaxa de deblocare a tabelelor

```
UNLOCK TABLES;
```

- Comenzile **LOCK** și **UNLOCK** nu trebuie folosite în tabelele InnoDB