

# MySQL

## • Partea V. Funcționalități MySQL (I)

Ce ne așteaptă?

1. Indexurile
2. Vederile
3. Procedurile stocate
4. Controlul fluxului

# 1. Indexurile

## Noțiunea de index

- **Indexul** – o structură de date ce permite creșterea vitezei de căutare a datelor într-un tabel
- În lipsa unui index, datele necesare se vor căuta prin examinarea rând pe rând a fiecărei înregistrări începând cu prima și până se găsesc respectivele date, acest lucru necesitând mult timp în cazul bazelor de date voluminoase
- Indexul permite reducerea timpului de căutare prin crearea unui tabel al indexului cu ordonarea datelor și căutarea datelor necesare folosind un algoritm de căutare (de exemplu algoritmul arbore binar)
- În tabelul index, datele coloanei pe care se creează indexul se aranjează în ordine alfabetică dacă sunt de tip string, în ordine numerică crescătoare dacă sunt de tip numeric și în ordine a datei timpului în cazul tipului datetime

## Tipuri de indexuri

- În funcție de strategie de căutare indexurile pot fi:
  - Pe o singură coloană – permite căutarea conform datelor unei singure coloane
  - Pe coloane multiple – permite căutarea conform combinațiilor datelor mai multor coloane
- În funcție de opțiunile datelor de căutare indexurile pot fi:
  - Index unic – toate datele coloanei sau combinația de date a mai multor coloane trebuie să fie unice
  - Index cheie primară – index unic ce nu acceptă valori lipsă (Not Null)
  - Index simplu (normal, regulat) – index pentru care valorile pot să nu fie unice și pot fi și lipsă
  - Index fulltext – index pentru date de tip text și se utilizează căutarea unui text ce conține o anumită secvență
  - Index descendent – index simplu cu ordonarea inversă, util pentru căutările în ultimele date introduse

## Avantajele și dezavantajele indexurilor

- **Avantajul indexurilor**
  - Creșterea vitezei de căutarea a datelor în cazul utilizării instrucțiunii **SELECT** cu diferite clauze (**WHERE**, **JOIN**, **ORDER BY**, etc)
- **Dezavantajele indexurilor**
  - Se creează un tabel a indexului deci se ocupă spațiu de memorie
  - Se reduce viteza instrucțiunilor **INSERT**, **UPDATE** și **DELETE** deoarece sunt necesare operațiile respective și în tabelul indexurilor
  - Riscul setării mai multor tipuri pe index pe una și aceeași coloana
- **Nu este rentabilă utilizarea indexurilor atunci când:**
  - Tabelul are puține înregistrări
  - Tabelul este preponderent utilizat pentru operații de înregistrare și actualizarea a datelor și mai puțin pentru operații de căutare a datelor

## Indexurile clustered și non-clustered

- Indexurile clustered sunt indexurile coloanelor cheii primare și cheii străine setați automat și care permit accesarea directă a datelor căutate
- Dacă nu se definește cheia primară sau străină atunci index clustered devine primul index unic fără lipsuri, iar dacă și acesta lipsește, MySQL generează un index intern ascuns
- Indexul non-clustered este indexul setat manual pe oricare coloana și permite determinarea unui pointer cu care mai apoi se accesează datele căutate
- Indexul non-clustered atribuie fiecărei date un pointer în tabelul original și creează un tabel al indexurilor în care într-o coloană se includ datele din coloana tabelului original ordonate (alfabetic sau numeric crescător), iar în altă coloană pointerii corespunzători
- După determinarea pointerului datei căutate în tabelul indexurilor, se accesează datele din tabelul original conform acestuia

## Exemplu de tabel al indexului non-clustered

- Tabelul original

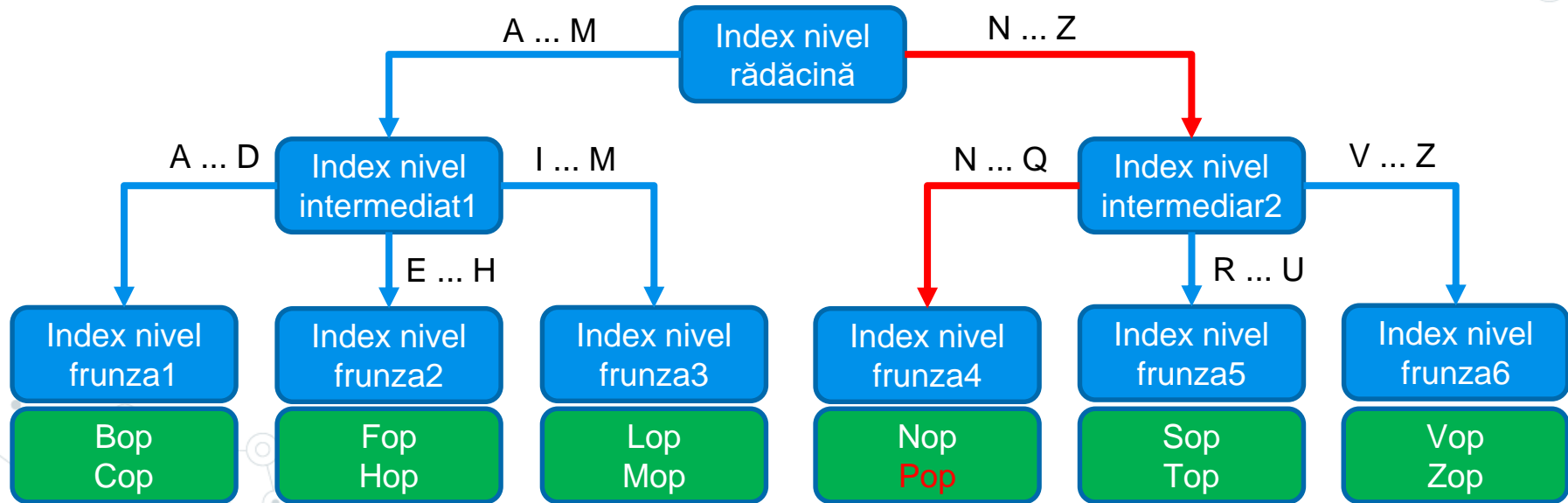
Id	Nume	Prenume	Vârstă	Pointer
1	Pop	Cab	34	P_1
2	Sop	Vab	23	P_2
3	Top	Dab	57	P_3
4	Lop	Sab	41	P_4
5	Cop	Mab	33	P_5
6	Vop	Nab	29	P_6
7	Nop	Fab	45	P_7
8	Mop	Hab	25	P_8
9	Bop	Lab	53	P_9
10	Hop	Bab	37	P_10
11	Fop	Tab	40	P_11
12	Zop	Gab	39	P_12

- Tabelul indexului pe coloana Nume

Pointer	Nume
P_9	Bop
P_5	Cop
P_11	Fop
P_10	Hop
P_4	Lop
P_8	Mop
P_7	Nop
P_1	Pop
P_2	Sop
P_3	Top
P_6	Vop
P_12	Zop

## Algoritmul arbore de căutare

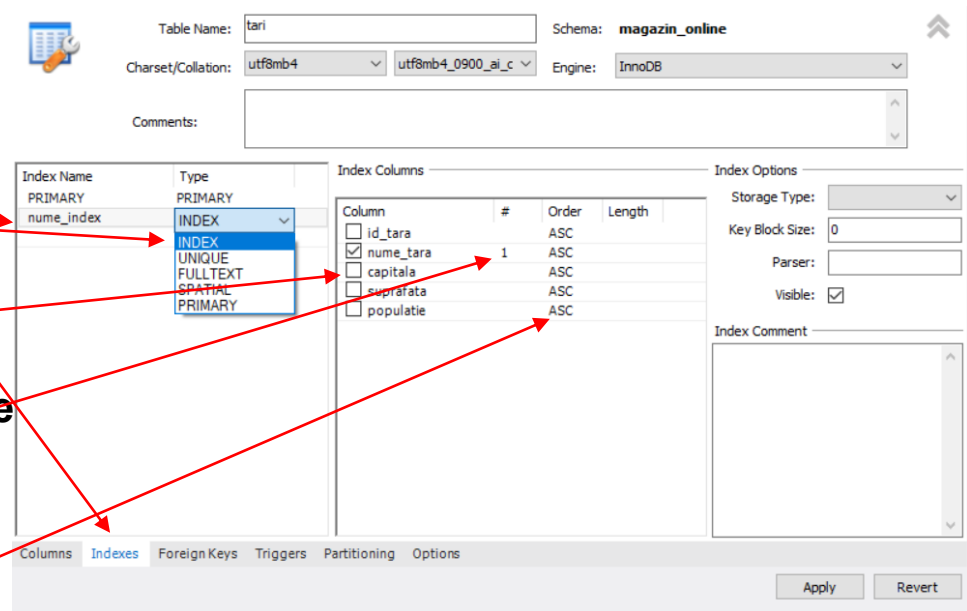
- Pentru căutarea numelui *Pop* se va examina calea la *indexul nivelului rădăcină*
- Deoarece litera p cu care începe numele se află în gama *N...Z*, în continuare se examinează *indexul nivel intermediar2*
- Deoarece litera p cu care începe numele se află în gama *N...Q*, în continuare se caută numele *Pop* în *lista indexului nivelului frunza4*





## Crearea indexului tabelului în MySQL Wordbench

- Se deschide fereastra de crearea/modificarea a tabelului
- Se selectează tab-ul Indexes
- Se specifică numele indexului în câmpul Index Name și se selectează tipul acestuia în câmpul Type
- Se bifează coloana/coloanele asupra cărora se aplică indexul
- În cazul indexului pe coloane multiple se urmărește ordinea bifării
- În cazul indexului descendent se modifică opțiunea ASC în DESC
- Pentru ștergerea indexului se apară clic dreapta pe numele acestuia în câmpul Index Name și se alege opțiunea Delete Selected
- Pentru generarea scriptului și executarea pe server se apasă butonul Apply

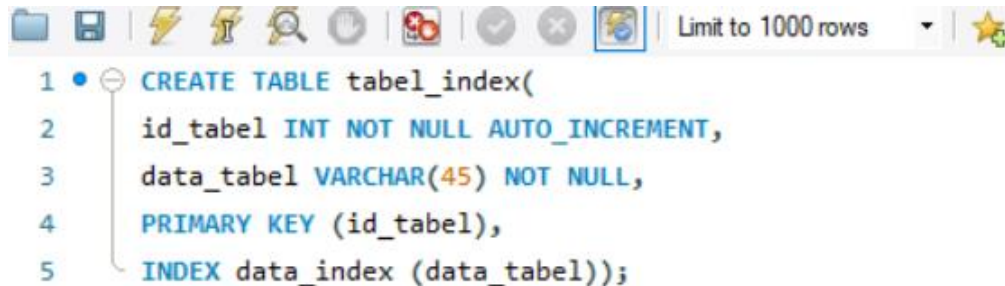


## Crearea indexului la crearea tabelului în SQL

- Crearea indexului odată cu crearea tabelului

```
CREATE TABLE nume_tabel(  
  Nume_col_1 tip_date optiuni  
  Nume_col_2 tip_date optiuni  
  INDEX/ FULLTEXT / UNIQUE INDEX (nume_col1_index, nume_col2_index ASC/DESC));
```

- Exemplu



The screenshot shows a SQL editor window with a toolbar at the top. The SQL code being entered is:

```
1 CREATE TABLE tabel_index(  
2   id_tabel INT NOT NULL AUTO_INCREMENT,  
3   data_tabel VARCHAR(45) NOT NULL,  
4   PRIMARY KEY (id_tabel),  
5   INDEX data_index (data_tabel));
```

Index Name	Type	Index Columns	Index
PRIMARY	PRIMARY		St
data_index	INDEX		Ke


Column	#	Order	Length
<input type="checkbox"/> id_tabel		ASC	
<input checked="" type="checkbox"/> data_tabel	1	ASC	

## Crearea indexului într-un tabel existent în SQL (1)

- Crearea directă a indexului într-un tabel existent

```
CREATE INDEX nume_index  
ON nume_tabel (nume_col1_index, nume_col2_index ASC/DESC));
```

- Exemplu



```
1 CREATE INDEX index_client ON client (nume_client, prenume_client);
```

Index Name	Type	Index Columns				Indi
PRIMARY	PRIMARY					S
fk_tara_id	INDEX					K
index_client	INDEX					

Column	#	Order	Length
<input type="checkbox"/> id		ASC	
<input checked="" type="checkbox"/> nume_client	1	ASC	
<input checked="" type="checkbox"/> prenume_client	2	ASC	
<input type="checkbox"/> tara		ASC	

## Crearea indexului într-un tabel existent în SQL (2)

- Crearea prin modificarea tabel existent

```
ALTER TABLE nume_tabel  
ADD INDEX nume_index (nume_col1_index, nume_col2_index ASC/DESC));
```

- Exemplu



1 ALTER TABLE client ADD FULLTEXT index\_num (nume\_client);

Index Name	Type	Index Columns				Index Op
PRIMARY	PRIMARY					Storag
fk_tara_id	INDEX					Key Blc
index_client	INDEX					
index_num	FULLTEXT					

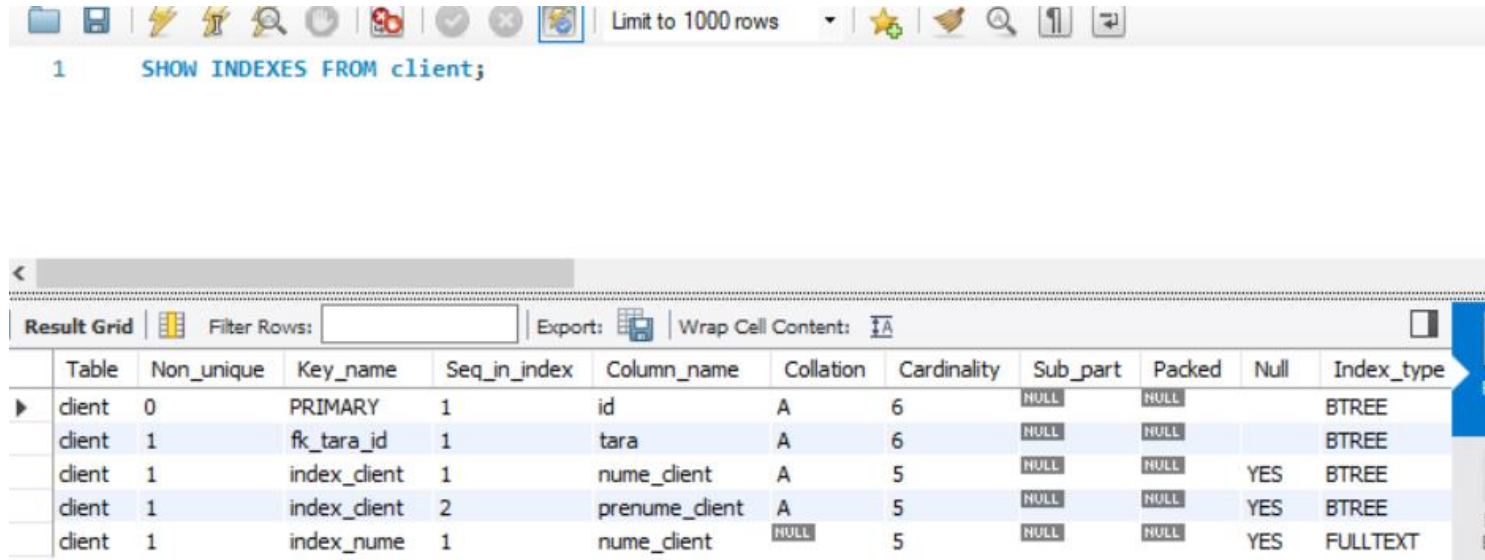
Column	#	Order	Length	
<input type="checkbox"/> id		ASC		
<input checked="" type="checkbox"/> nume_client	1	ASC		
<input type="checkbox"/> prenume_client		ASC		
<input type="checkbox"/> tara		ASC		

## Vizualizarea indexurilor unui tabel în SQL

- Vizualizarea indexurilor unui tabel

SHOW INDEXES FROM nume\_tabel;

- Exemplu



The screenshot shows a SQL client interface with a toolbar at the top. Below the toolbar, the command `SHOW INDEXES FROM client;` is entered in a text area. The results are displayed in a table with the following columns: Table, Non\_unique, Key\_name, Seq\_in\_index, Column\_name, Collation, Cardinality, Sub\_part, Packed, Null, and Index\_type.

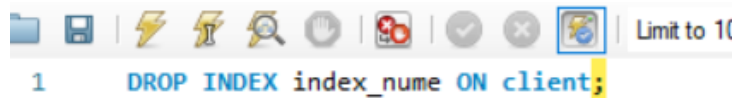
Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
client	0	PRIMARY	1	id	A	6	NULL	NULL		BTREE
client	1	fk_tara_id	1	tara	A	6	NULL	NULL		BTREE
client	1	index_client	1	nume_client	A	5	NULL	NULL	YES	BTREE
client	1	index_client	2	prenume_client	A	5	NULL	NULL	YES	BTREE
client	1	index_num	1	nume_client	NULL	5	NULL	NULL	YES	FULLTEXT

## Ștergerea unui index din tabel în SQL

- Ștergerea directă a unui index al tabelului

```
DROP INDEX nume_index ON nume_tabel;
```

- Exemplu



- Ștergerea indexului prin modificarea tabelului

```
ALTER TABLE nume_tabel DROP INDEX nume_index;
```

- Exemplu



## 2. Vederile

### Noțiunea de vedere

- **Vederea** – reprezintă un tabel virtual ce conține o combinație de coloane ce aparțin mai multor tabele reale
- Asupra vederilor se pot efectua interogările SELECT, iar în funcție de dreptul utilizatorului și interogările INSERT, UPDATE și DELETE
- Necesitatea interogărilor reiese din:
  - Securitatea datelor – prin intermediul vederilor se poate face acces doar la unele coloane ale tabelelor, celelalte rămânând securizate
  - Confortul – pentru a vizualiza datele complete nu mai este necesară accesarea mai multor tabele, ci pot fi setate spre vizualizarea într-o singură vedere

## Crearea vederii

- Crearea unei vedere se realizează cu interogarea

```
CREATE VIEW nume_vedere AS interogare_selectie
```

- Pentru a evita erorile in cazul existentei unei vedere cu acest nume se va utiliza

```
CREATE OR REPLACE VIEW nume_vedere AS interogare_selectie
```

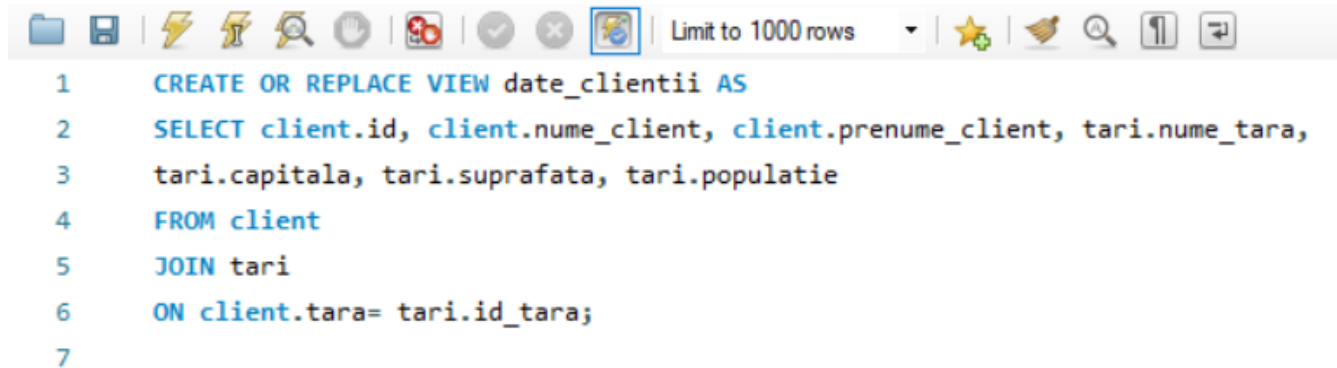
- Interogarea de selecție reprezintă o interogarea **SELECT** cu condiții **WHERE** sau cu **JOIN** în cazul coloanelor din mai multe tabele

```
SELECT nume_tabel_1.nume_col, nume_tabel_2.nume_col, nume_tabel_3.nume_col  
FROM nume_tabel_1  
JOIN nume_tabel_2  
ON nume_tabel_1.nume_col_cheie_strain= nume_tabel_2.nume_col_cheie_primar  
JOIN nume_tabel_3  
ON nume_tabel_2.nume_col_cheie_strin= nume_tabel_3.nume_col_cheie_primar;
```



## Exemplu de crearea a vederii

- Elaborarea vederii de vizualizarea a datelor despre numele clienților, prenumele, țara de origine, capitala, suprafața și populația țării în baza magazinului online



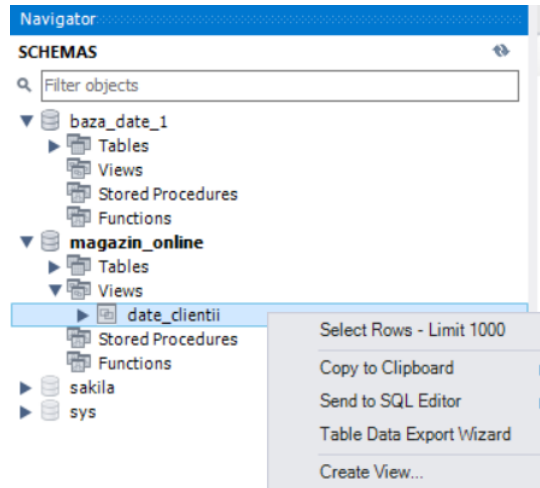
```
1 CREATE OR REPLACE VIEW date_clientii AS
2 SELECT client.id, client.num_e_client, client.prenume_client, tari.num_e_tara,
3    tari.capitala, tari.suprafata, tari.populatie
4 FROM client
5 JOIN tari
6 ON client.tara= tari.id_tara;
7
```

## Vizualizarea vederii

- Pentru vizualizarea tuturor datelor vederii se va scrie interogarea

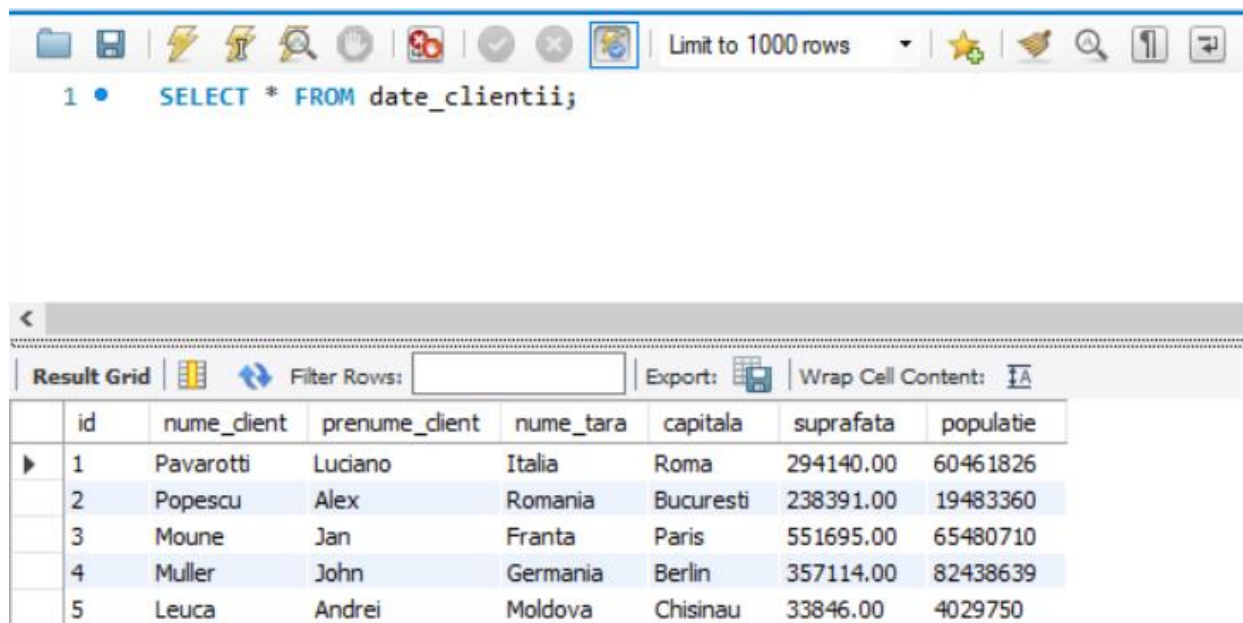
```
SELECT * FROM nume_vedere;
```

- În Workbench pentru vizualizarea vederii se va aplica clic drept numele vederii în secțiunea Views a câmpului de navigarea și apoi se selectează Select Rows – Limit 100



## Exemplu de vizualizarea a vederii

- Vizualizarea vederii create date\_clienti



The screenshot shows a MySQL database client interface. At the top, there is a toolbar with various icons for file operations, execution, and search. Below the toolbar, a text area contains the SQL query: `1 • SELECT * FROM date_clientii;`. Below the query, there is a section for the query results. It includes a "Result Grid" tab, a "Filter Rows" input field, an "Export" button, and a "Wrap Cell Content" checkbox. The results are displayed in a table with 8 columns: `id`, `nume_client`, `prenume_client`, `nume_tara`, `capitala`, `suprafata`, and `populatie`. The table contains 5 rows of data.

	id	nume_client	prenume_client	nume_tara	capitala	suprafata	populatie
▶	1	Pavarotti	Luciano	Italia	Roma	294140.00	60461826
	2	Popescu	Alex	Romania	Bucuresti	238391.00	19483360
	3	Moune	Jan	Franta	Paris	551695.00	65480710
	4	Muller	John	Germania	Berlin	357114.00	82438639
	5	Leuca	Andrei	Moldova	Chisinau	33846.00	4029750

## Modificarea și ștergerea vederilor

- Pentru modificarea unei vederi se va actualiza interogarea de selecție prin intermediul interogarii ALTER

```
ALTER VIEW nume_vedere AS interogare_selectie_noua;
```

- Pentru ștergerea vederii se va executa interogarea

```
DROP VIEW nume_vedere;
```

- Exemplu de ștergere:



### 3. Procedurile stocate

#### Noțiune de procedură stocată

- **Procedurile stocate** sunt blocuri de cod SQL stocate pe server ce pot efectua diferite activități pe server.
- Procedurile stocate acceptă parametri de intrare și de ieșire, pot returna mai multe valori și pot fi apelate de mai multe ori în diferite situații
- O procedură stocată poate apela alte proceduri stocate sau funcții
- Avantajele procedurilor stocate:
  - Reducerea traficului - mai multe interogări SQL sunt încapsulate în procedură și în locul interogărilor multiple se transmite doar numele procedurii și parametrii ei
  - Simplitatea mentenanței – procedurile stocate sunt reutilizabile. Diferite aplicații pot utiliza aceste proceduri iar dacă sunt necesare unele modificări acestea se realizează doar în proceduri
  - Securitatea – procedurile stocate sunt mai securizate decât interogările multiple deoarece se poate defini utilizatorul ce va avea acces la acestea

## Crearea și apelarea procedurilor

- Pentru crearea unei proceduri se va utiliza sintaxa

```
DELIMITER //  
CREATE PROCEDURE nume_procedura (param1, param2, param3)  
BEGIN  
  corp_procedura  
END//  
DELIMITER ;
```

- Parametrii procedurii sunt opționali
- Corpul procedurii conține mai multe interogări separate prin ;
- Pentru a delimita procedura stocat în cadrul SQL se modifică delimitatorul dintre interogări din ; în // iar apoi se revine la delimitatorul implicit ;
- Pentru apelarea unei proceduri se va utiliza sintaxa

```
CALL nume_procedura (param1, param2, param3)
```

- Parametrii la apelarea procedurii sunt necesari dacă au fost definiți în procedură

## Exemplu de creare a procedurii

- Crearea procedurii `populatie_tari` de accesare a numelui țării și populației acesteia conform datelor tabelului `tari`

```
1 DELIMITER //
```

```
2 CREATE PROCEDURE populatie_tari ()
```

```
3 BEGIN
```

```
4     SELECT nume_tara, populatie FROM tari ;
```

```
5 END//
```

```
6 DELIMITER ;
```

- Apelarea procedurii `populatie_tari`

```
1 CALL populatie_tari();
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
nume_tara	populatie			
Italia	60461826			
Romania	19483360			
Franta	65480710			
Germania	82438639			
Moldova	4029750			

## Parametrii procedurilor stocate

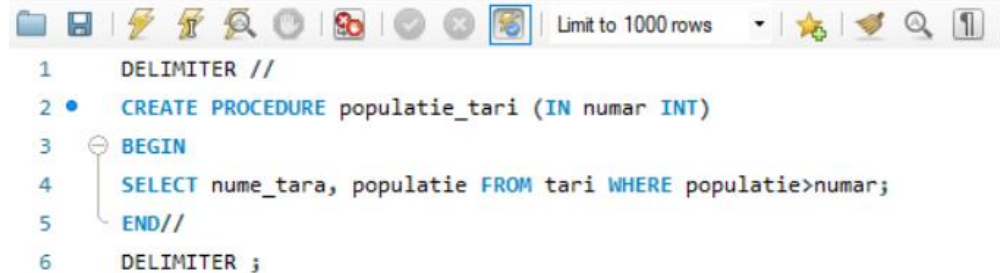
- **Procedurile stocate pot avea 3 moduri de parametri:**
  - **IN** – parametru de intrare, se utilizează atunci când procedura accepta parametri de intrare dar nu returnează nimic
  - **OUT** – parametru de ieșire, se utilizează atunci când procedura nu are parametri de intrare dar returnează rezultate generate de interogările SQL din corpul acesteia
  - **INOUT** – parametru de ieșire-ieșire, este însoțit și de un parametru de intrare și se utilizează atunci când procedura are parametri de intrare și returnează rezultate bazate pe acest parametru.
- **Sintaxa definirii parametrilor**

```
CREATE PROCEDURE nume_procedura  
(IN/OUT/INOUT nume_param_1 tip_param_1, IN/OUT/INOUT nume_param_2 tip_param_2)  
BEGIN  
corp_procedura  
END
```



## Exemplu de crearea a procedurii cu parametri IN

- Crearea procedurii `populatie_tari` de accesare a numelui țării și populației acesteia dacă populația este mai mare de un anumit număr

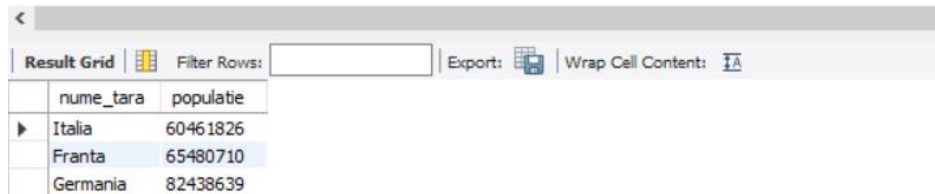


```
1 DELIMITER //
2 CREATE PROCEDURE populatie_tari (IN numar INT)
3 BEGIN
4 SELECT nume_tara, populatie FROM tari WHERE populatie>numar;
5 END//
6 DELIMITER ;
```

- Apelarea procedurii `populatie_tari` cu parametrul `numar = 20000000`



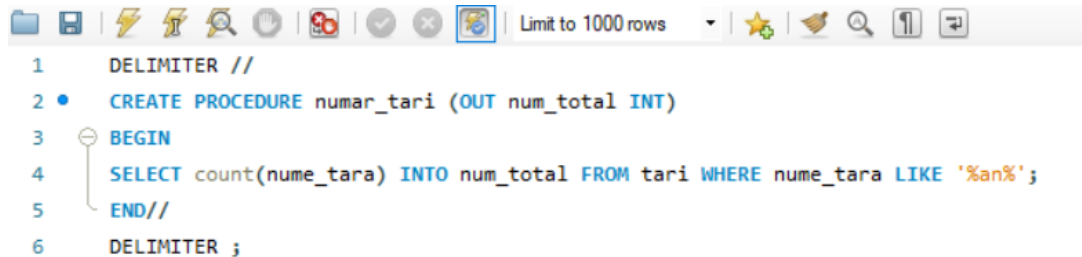
```
1 CALL populatie_tari(20000000);
```



	nume_tara	populatie
▶	Italia	60461826
	Franta	65480710
	Germania	82438639

## Exemplu de crearea a procedurii cu parametri OUT

- Crearea procedurii `numar_tari` care returnează numărul de țări ce au în componența numelui secvența an



```
1 DELIMITER //
```

```
2 • CREATE PROCEDURE numar_tari (OUT num_total INT)
```

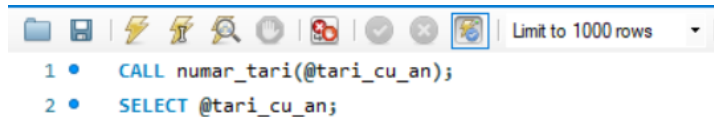
```
3 BEGIN
```

```
4 SELECT count(ume_tara) INTO num_total FROM tari WHERE ume_tara LIKE '%an%';
```

```
5 END//
```

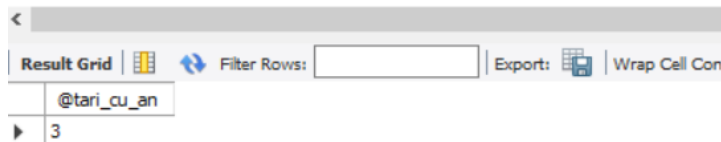
```
6 DELIMITER ;
```

- Apelarea procedurii `numar_tari` prin trecerea unei variabilei ca parametru al procedurii, variabilă căreia i se va atribui rezultatul returnat



```
1 • CALL numar_tari(@tari_cu_an);
```

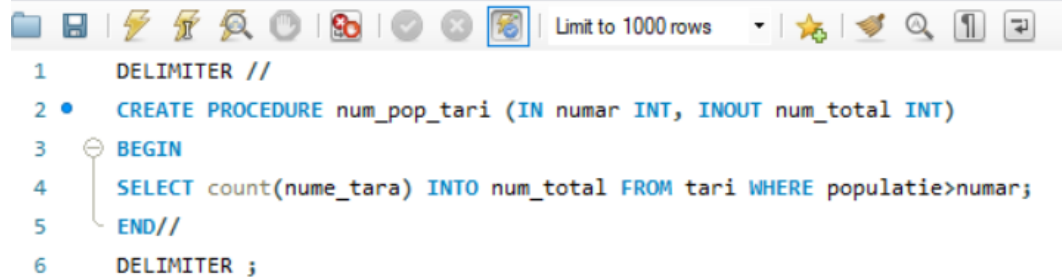
```
2 • SELECT @tari_cu_an;
```



@tari_cu_an
3

## Exemplu de crearea a procedurii cu parametri INOUT

- Crearea procedurii num\_pop\_tari de afișarea a numărului de țări pentru care populația este mai mare de un anumit număr



```
1 DELIMITER //
2 • CREATE PROCEDURE num_pop_tari (IN numar INT, INOUT num_total INT)
3 BEGIN
4 SELECT count(ume_tara) INTO num_total FROM tari WHERE populatie>numar;
5 END//
6 DELIMITER ;
```

- Apelarea procedurii num\_pop\_tari



```
1 • CALL num_pop_tari(10000000, @numarul_tarilor);
2 • SELECT @numarul_tarilor;
```



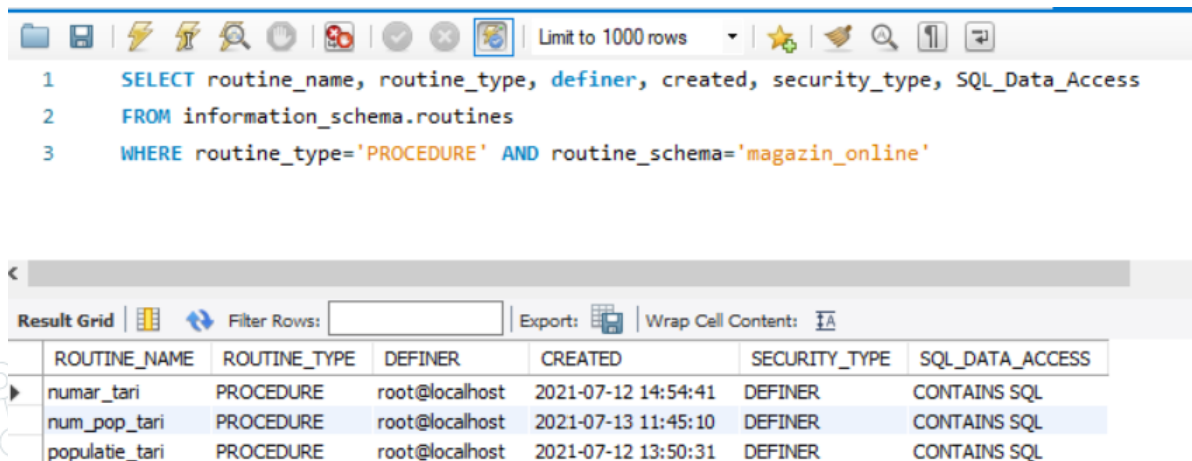
Result Grid		Filter Rows:	Export:	Wrap Cell Conten
	@numarul_tarilor			
▶	4			

## Vizualizarea informației despre procedurile unei baze

- Pentru vizualizarea listei procedurilor și a informațiilor despre acestea se utilizează sintaxa:

```
SELECT routine_name, routine_type, definer, created, security_type, SQL_Data_Access  
FROM information_schema.routines  
WHERE routine_type='PROCEDURE' AND routine_schema=nume_baza'
```

- Exemplu



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

```
1 SELECT routine_name, routine_type, definer, created, security_type, SQL_Data_Access  
2 FROM information_schema.routines  
3 WHERE routine_type='PROCEDURE' AND routine_schema='magazin_online'
```

Below the query, the results are displayed in a table with the following columns: ROUTINE\_NAME, ROUTINE\_TYPE, DEFINER, CREATED, SECURITY\_TYPE, and SQL\_DATA\_ACCESS. The table contains three rows of data.

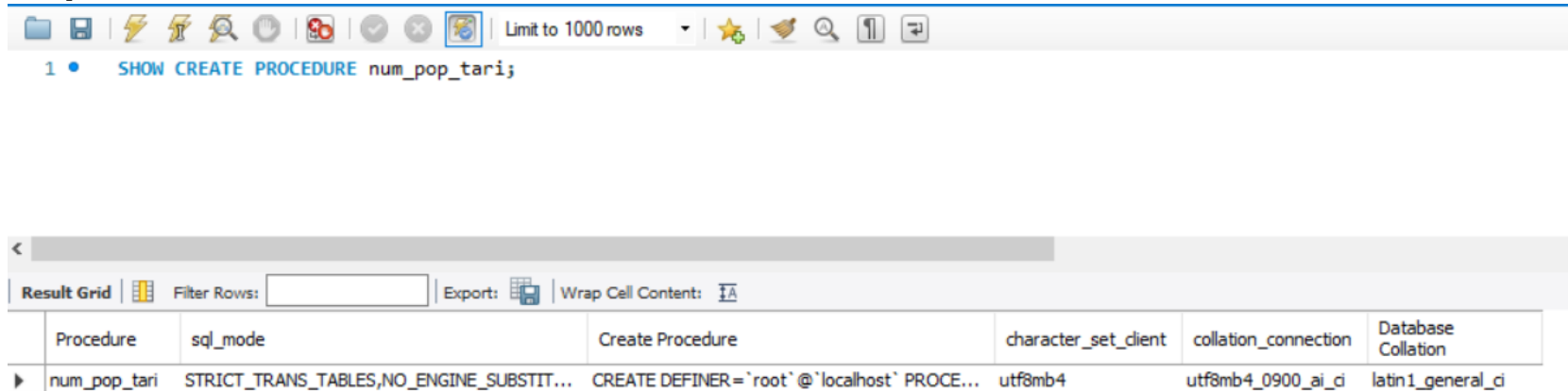
ROUTINE_NAME	ROUTINE_TYPE	DEFINER	CREATED	SECURITY_TYPE	SQL_DATA_ACCESS
numar_tari	PROCEDURE	root@localhost	2021-07-12 14:54:41	DEFINER	CONTAINS SQL
num_pop_tari	PROCEDURE	root@localhost	2021-07-13 11:45:10	DEFINER	CONTAINS SQL
populatie_tari	PROCEDURE	root@localhost	2021-07-12 13:50:31	DEFINER	CONTAINS SQL

## Vizualizarea și ștergerea unei proceduri

- Vizualizarea unei proceduri create

`SHOW CREATE PROCEDURE nume_procedură;`

- Exemplu



The screenshot shows a MySQL client window with the command `SHOW CREATE PROCEDURE num_pop_tari;` entered in the query editor. Below the editor, the results are displayed in a table with the following columns: Procedure, sql\_mode, Create Procedure, character\_set\_client, collation\_connection, and Database Collation. The table contains one row of data for the procedure `num_pop_tari`.

Procedure	sql_mode	Create Procedure	character_set_client	collation_connection	Database Collation
num_pop_tari	STRICT_TRANS_TABLES,NO_ENGINE_SUBSTIT...	CREATE DEFINER='root'@'localhost' PROCE...	utf8mb4	utf8mb4_0900_ai_ci	latin1_general_ci

- Ștergerea unei proceduri create

`DROP PROCEDURE IF EXISTS nume_procedură;`

## 4. Controlul fluxului

### Eșența controlului fluxului

- **Controlul fluxului – utilizarea unor mecanisme de creare a ramificațiilor sau buclelor în ordinea executării interogărilor**
- **Controlul fluxului este posibil doar în corpul procedurilor stocare sau a funcțiilor definite de utilizator**
- **Cele mai des utilizate declarații în controlul fluxului sunt:**
  - **IF-THEN-ELSE**
  - **CASE**
  - **WHILE**
  - **LOOP**
  - **REPEAT**

## Ramificația IF-THEN-ELSE

- **IF-THEN-ELSE** se utilizează pentru executarea unui cod dacă condiția este adevărată sau un alt cod dacă condiția este falsă

- **Sintaxa IF-THEN-ELSE**

```
IF conditie_1 THEN  
    cod_1_executat  
ELSEIF conditie_2 THEN  
    cod_2_executat  
ELSE  
    cod_3_executat  
END IF;
```

- **Declarațiile ELSEIF și ELSE sunt opționale**
- **Pot exista mai multe declarații ELSEIF cu diferite condiții**
- **Se executa primul cod a cărui condiție este adevărată în ordinea scrierii lor sau codul ELSE dacă nici o condiție nu este adevărată**

## Exemplu utilizarea IF-ELSE-THEN

```

1  DELIMITER //
2  CREATE PROCEDURE populatie_tari (IN numar INT)
3  BEGIN
4  IF numar<10000000 THEN
5      SELECT nume_tara, populatie FROM tari WHERE nume_tara='Moldova';
6  ELSEIF numar>10000000 AND numar<20000000 THEN
7      SELECT nume_tara, populatie FROM tari WHERE nume_tara='Romania';
8  ELSE
9      SELECT nume_tara, populatie FROM tari WHERE populatie>numar;
10 END IF;
11 END//
12 DELIMITER ;

```

```

1  CALL populatie_tari(5000000);

```

```

1  CALL populatie_tari(15000000);

```

```

1  CALL populatie_tari(25000000);

```

Result Grid		Filter Rows:	Export
nume_tara	populatie		
Moldova	4029750		

Result Grid		Filter Rows:	Export
nume_tara	populatie		
Romania	19483360		

Result Grid		Filter Rows:	Export
nume_tara	populatie		
Italia	60461826		
Franta	65480710		
Germania	82438639		



## Ramificația CASE

- **CASE** se utilizează identic **IF** cu mai multe **ELSEIF**
- **CASE** are 2 sintaxe:
  - După **CASE** urmează o expresie iar după **WHEN** o valoarea cu care se compara expresia
  - După **CASE** nu urmează nimic iar după **WHEN** o condiție care se verifică

- **Sintaxa 1 CASE**

```
CASE expresie
  WHEN valoarea_1 THEN
    cod_1_executat
  WHEN valoarea_2 THEN
    cod_2_executat
  ELSE
    cod_3_executat
END CASE;
```

- **Sintaxa 2 CASE**

```
CASE
  WHEN conditie_1 THEN
    cod_1_executat
  WHEN conditie_2 THEN
    cod_2_executat
  ELSE
    cod_3_executat
END CASE;
```

- Sunt obligatorii declarația **CASE** și prima declarație **WHEN+THEN**

## Exemplu utilizarea CASE

```
1 DELIMITER //
```

```
2 • CREATE PROCEDURE populatie_tari_2 (IN numar INT)
```

```
3 BEGIN
```

```
4 CASE
```

```
5     WHEN numar<10000000 THEN
```

```
6         SELECT nume_tara, populatie FROM tari WHERE nume_tara='Moldova';
```

```
7     WHEN numar>10000000 AND numar<20000000 THEN
```

```
8         SELECT nume_tara, populatie FROM tari WHERE nume_tara='Romania';
```

```
9     ELSE
```

```
10        SELECT nume_tara, populatie FROM tari WHERE populatie>numar;
```

```
11 END CASE;
```

```
12 END//
```

```
13 DELIMITER ;
```

```
1 CALL populatie_tari_2(5000000);
```

Result Grid		Filter Rows:	Export
nume_tara	populatie		
Moldova	4029750		

```
1 CALL populatie_tari_2(15000000);
```

Result Grid		Filter Rows:	Export
nume_tara	populatie		
Romania	19483360		

```
1 CALL populatie_tari_2(25000000);
```

Result Grid		Filter Rows:	Export
nume_tara	populatie		
Italia	60461826		
Franta	65480710		
Germania	82438639		

## Buclo WHILE

- **WHILE** se utilizează pentru a se executa repetat un cod dar nu se știe de câte ori se va executa sau dacă se va executa vreo dată
- **Sintaxa WHILE**

```
nume_buclo: WHILE conditie  
DO cod_executat  
END WHILE nume_buclo;
```

- **Numele buclei este opțional**
- **Codul declarației DO se va executa dacă condiția WHILE este adevărată**
- **Pentru a evita formarea buclor infinite în codul declarației DO trebuie introdus un mecanism de modificarea a condiției**

## Exemplu utilizarea WHILE

```
1 DELIMITER //
```

```
2 • CREATE PROCEDURE numarare (IN numar INT)
```

```
3 BEGIN
```

```
4     SET @numar_bucle=0;
```

```
5     bucla_numarare: WHILE numar<5
```

```
6     DO
```

```
7         SET @numar_bucle=@numar_bucle+1;
```

```
8         SET numar=numar+1;
```

```
9     END WHILE bucla_numarare;
```

```
10    SELECT @numar_bucle;
```

```
11 END//
```

```
12 DELIMITER ;
```

1 CALL numarare (0)

@numar_bucle
5

1 CALL numarare (3)

@numar_bucle
2

1 CALL numarare (6)

@numar_bucle
0

## Buclo LOOP + LEAVE +ITERATE

- **LOOP** se utilizează pentru a se executa repetat un cod dar nu se știe de câte ori se va executa dar se va executa cel puțin odată
- Pentru trecerea la următoarea iterație se utilizează declarația **ITERATE**
- Pentru ieșirea din buclă se utilizează declarația **LEAVE**
- **Sintaxa WHILE**

```
nume_buclo: LOOP  
    cod_executat;  
    IF conditie THEN  
        ITERATE nume_buclo;  
    END IF;  
    LEAVE nume_buclo;  
END LOOP nume_buclo;
```

- Declarațiile **ITERATE** și **LEAVE** pot fi condiționate cu ajutorul unei condiții **IF**

## Exemplu utilizarea LOOP + LEAVE +ITERATE

```
1 DELIMITER //
```

```
2 • CREATE PROCEDURE numarare_2 (IN numar_1 INT)
```

```
3 BEGIN
```

```
4   SET @numar_2 = 0;
```

```
5   bucla_numarare: LOOP
```

```
6     SET @numar_2=@numar_2+numar_1;
```

```
7     IF @numar_2<=10 THEN
```

```
8       ITERATE bucla_numarare;
```

```
9     END IF;
```

```
10    LEAVE bucla_numarare;
```

```
11  END LOOP bucla_numarare;
```

```
12  SELECT @numar_2;
```

```
13 END//
```

```
14 DELIMITER ;
```

```
1 CALL numarare_2 (1)
```

Result Grid










	@numar_2
▶	11

## Buclo REPEAT










- **REPEAT** se utilizează pentru a se executa repetat un cod până la îndeplinirea unei condiții dar se va executa cel puțin odată
- Pentru introducerea condiției se utilizează declarație **UNTIL**
- **Sintaxa WHILE**

```
nume_buclo: REPEAT  
    cod_executat;  
    UNTIL condiție  
END REPEAT nume_buclo;
```

## Exemplu utilizarea REPEAT

         Limit to 1000 rows  

```
1 DELIMITER //  
2 CREATE PROCEDURE bucla_repeat (IN numar INT)  
3 BEGIN  
4 SET @numar_buc1a=0;  
5 bucla_repeat: REPEAT  
6 SET @numar_buc1a=@numar_buc1a+numar;  
7 UNTIL @numar_buc1a>101  
8 END REPEAT bucla_repeat;  
9 SELECT @numar_buc1a;  
10 END//  
11 DELIMITER ;
```


          

```
1 CALL bucla_repeat (11);
```

<

Result Grid |  Filter Rows:  | Export

	@numar_buc1a
▶	110