7. VIEW-URI, VARIABILE, PREPARED STATEMENTS, TRANZACTII

7.1. View-uri....2
7.1.1. Conceptul de view....2
7.1.2. Definirea unui view....2
7.1.2.1. Sintaxa generala....2
7.1.2.2. Numele unui view....3
7.1.2.3. Algoritmul unui view....3
7.1.2.4. Lista de coloane a unui view....4
7.1.2.5. Restrictii la definirea unui view....5
7.1.3. View-uri actualizabile....5
7.1.3.1. Conditii si tipuri de actualizare....5
7.1.3.2. Mentinerea consistentei datelor....6
7.1.4. Modificarea unui view....7
7.1.5. Stergerea unui view....7
7.1.6. Vizualizarea detaliilor unui view....7
7.1.7. Vizualizarea listei de view-uri....8
7.1.8. Verificarea unui view....8
7.2. Variabile definite de client (“user variables”)....9
7.2.1. Concepte si utilitate....9
7.2.2. Numele unei variabile....9
7.2.3. Tipul de date al unei variabile....9
7.2.4. Definirea unei variabile....9
7.2.5. Utilizarea si domeniul de vizibilitate al unei variabile....10
7.2.6. Alte tipuri de variabile....10
7.3. Prepared statements (interogari precompilate)....11
7.3.1. Ce este un prepared statement si avantajele sale....11
7.3.2. Crearea unui prepared statement....12
7.3.3. Executarea unui prepared statement....12
7.3.4. Stergerea unui prepared statement....13
7.4. Tranzactii....13
7.4.1. Conceptul de tranzactie....13
7.4.2. Tranzactiile si modul autocommit....14
7.4.3. Incheierea unei tranzactii....15
7.5. BIBLIOGRAFIE....15

InfoAcademy

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

7.1. View-uri

7.1.1. Conceptul de view

Un view reprezinta o interogare de tip SELECT a carei definitie este stocata in baza de date si care produce un set de rezultate. In acest fel, view-ul actioneaza ca o tabela virtuala:

- din el se pot extrage date
- asupra lui se pot stabili privilegii
- in anumite conditii view-ul poate fi actualizabil, putandu-i-se aplica celelalte instructiuni DML (INSERT, UPDATE, DELETE)

Ceea ce diferentiaza un view de o tabela reala este ca el nu contine datele, ci specifica doar modul de a le obtine.

Pornind de la datele deja stocate in tabele, un view are un numar de libertati:

- poate selecta date dintr-o singura tabela sau din mai multe (inclusiv din alte view-uri!)
- poate aplica prelucrari asupra datelor si poate agrega valorile obtinute in caz de nevoie
- poate furniza doar un subset al coloanelor din tabela/tabelele de provenienta a datelor, selectand doar coloanele dorite in definitia view-ului
- poate furniza doar un subset al randurilor din tabela/tabelele de provenienta, aplicand clauza WHERE instructiunii SELECT ce defineste view-ul

Utilizatorul view-ului va primi datele “de-a gata”, deja filtrate/prelucrate, fara a avea nevoie sa efectueze el acele prelucrari/filtrari/join-uri.

Iata cateva utilizari posibile ale view-urilor:

- memorarea in baza de date a interogarilor efectuate des si ale caror rezultate participa eventual in alte interogari (ex: sumare, rapoarte)
- uneori dorim sa prezentam anumitor clienti numai anumite informatii din baza de date – fie din motive de securitate (pentru a nu le oferi acces la date confidentiale sau pentru a nu risca ca ei sa modifice date din greseala), fie din motive de usurinta (clientul are nevoie numai de o parte a informatiei din tabela/baza de date). Putem crea un view care selecteaza numai coloanele de interes/accesibile clientului, stabilind privilegiile de asa natura incat clientul sa nu poata accesa decat coloanele dorite
- la schimbarea structurii unei baze de date deja folosite, pentru a nu impieta asupra functionarii aplicatiilor care folosesc vechea structura, putem crea view-uri care intretin pentru aceste aplicatii iluzia ca baza de date are inca structura veche. In acest fel castigam timpul necesar migrarii treptate a aplicatiilor la noua structura a bazei de date

7.1.2. Definirea unui view

7.1.2.1. Sintaxa generala

Definirea unui VIEW se efectueaza cu ajutorul instructiunii CREATE VIEW, care are sintaxa urmatoare:

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    VIEW numeview [(nume_coloana_1, nume_coloana_2,....)]
    AS instructiune_select
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

In forma sa cea mai simpla, definirea unui view poate fi efectuata astfel:

```
CREATE VIEW numeview AS instructiune_select
```

Exemplul 1: un view care produce o singura inregistrare, continand o varsta aleatoare, o inaltime aleatoare si o greutate aleatoare (la fiecare citire din acest view se va obtine alt set de valori):

```
CREATE VIEW RandomPersonData (Varsta, Inaltime, Greutate) AS
      SELECT
            FLOOR(RAND()*120),
            FLOOR(50+RAND()*200),
            FLOOR(1+RAND()*300);
```

Exemplul 2: data fiind o tabela *Produse* cu coloanele *NumeProdus* si *Pret*, putem crea un view care arata doar produsele al caror pret se situeaza peste un anumit prag, interogand apoi acest view ca pe o tabela obisnuita:

```
-- Definire view
CREATE VIEW ProduseScumpe AS SELECT * FROM Produse WHERE Pret>1000;

-- Utilizare view
SELECT NumeProdus, Pret*1.24 AS PretCuTVA FROM ProduseScumpe WHERE NumeProdus LIKE '%swarowski%';
```

Exemplul 3: un view care produce pretul maxim si minim per categorie, agregand datele din doua tabele – *Categorii* si *Produse*:

```
CREATE VIEW PreturiExtremePerCategorie AS
      SELECT Categorie, MIN(Pret) AS PretMinim, MAX(Pret) AS PretMaxim
      FROM Produse INNER JOIN Categorii ON c_id = p_c_id GROUP BY Categorie
```

Vom detalia in continuare restrictiile si clauzele suplimentare implicate in crearea unui view.

7.1.2.2. Numele unui view

Un view este asociat unei baze de date – cu sintaxa de mai sus, el este creat automat in baza de date curenta. Pentru a il crea in alta baza de date este necesara specificarea numelui complet sub forma *bazadedate.numeview*.

View-urile impart acelasi spatiu de nume cu tabelele bazei de date in care sunt create, de aceea la definirea unui view nu trebuie sa existe deja o tabela cu acelasi nume.

In cazul in care in baza de date exista deja un view cu acelasi nume, clauza OR REPLACE va determina inlocuirea definitiei acestuia cu noua definitie. Clauza nu are insa efect daca in baza de date se gaseste o tabela cu numele respectiv – in acest caz, crearea view-ului va esua.

7.1.2.3. Algoritmul unui view

Clauza optionala **ALGORITHM** stabileste modul in care MySQL va procesa view-ul la accesarea acestuia de catre clienti. Valorile posibile sunt:

- **TEMPTABLE** – MySQL va crea o tabela temporara in care memoreaza datele corespunzatoare view-ului si va folosi acea tabela in operatiile in care este implicat view-ul. In acest fel, view-ul se comporta ca o tabela in toata regula, insa una read-only – ea contine o copie a datelor extrase din alte tabele, si in consecinta modificarea datelor din view nu va produce modificari in tabelele de provenienta

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

- **MERGE** – MySQL va procesa view-ul prin inlocuirea unor parti ale interogarii formulate de client cu parti ale interogarii ce defineste view-ul. Spre exemplu, daca consideram view-ul *ProduseScumpe* definit mai sus, interogarea ce-i urmeaza in cadrul exemplului va fi procesata astfel: numele view-ului va fi inlocuit cu numele tabelei, iar conditia de definitie a view-ului (Pret>1000) va fi adaugata la conditiile interogarii furnizate de client, rezultand interogarea:

```
SELECT NumeProdus,Pret*1.24 AS PretCuTVA From Produse
       WHERE pret>1000 AND NumeProdus LIKE '%swarowski%'
```

  ATENTIE! Algoritmul MERGE poate fi folosit numai atunci cand fiecare inregistrare din view corespunde exact unei inregistrari dintr-o tabela de baza. Aceasta exclude folosirea in definitia unui view de tip MERGE a urmatoarelor elemente:
  - functii de agregare
  - DISTINCT
  - GROUP BY
  - HAVING
  - LIMIT
  - UNION
  - subinterogari in FROM (in locul tabelei)
  - definirea view-ului pe baza unei interogari care nu selecteaza dintr-o tabela (ex: produce doar valori aleatoare, ca intr-unul din exemplele de mai sus)

- **UNDEFINED** – MySQL va alege singur algoritmul: el va prefera MERGE ori de cate ori este posibil, recurgand la solutia TEMPTABLE numai atunci cand nu are incotro (deoarece TEMPTABLE este mai ineficient, si in plus face view-ul neactualizabil). UNDEFINED este valoarea implicita atunci cand clauza ALGORITHM lipseste.

7.1.2.4. Lista de coloane a unui view

Numele de coloane ale unui view trebuie sa fie unice in cadrul view-ului. Numele de coloane implicite produse de view sunt cele returnate de interogarea SELECT ce-l defineste. Atunci cand acestea contin duplicate sau, din alte motive, nu sunt convenabile, putem lua urmatoarele masuri:

- folosirea clauzei AS in cadrul intructiunii SELECT pentru a redenumi coloanele “buclucase”:

```
-- daca coloana nume se regaseste in ambele tabele ale unui join:
CREATE VIEW CategoriiProduse AS
       SELECT Categorii.Nume AS NumeCategorie,Produse.Nume AS NumeProdus
       FROM Produse INNER JOIN Categorii ON (c_id = p_c_id)
```

- specificarea numelui tuturor coloanelor produse de view, specificandu-le intre paranteze imediat dupa numele view-ului in definitia acestuia. Numarul de nume de coloane specificate in definitia view-ului trebuie sa fie egal cu numarul de coloane produse de interogarea SELECT ce-l defineste:

```
-- solutie alternativa:
CREATE VIEW CategoriiProduse(NumeCategorie, NumeProdus) AS
       SELECT Categorii.Nume,Produse.Nume FROM Produse INNER JOIN Categorii
       ON (c_id = p_c_id)
```

ATENTIE! Lista de coloane a unui view este “batuta in cuie” la crearea view-ului, chiar daca definitia tabelelor participante se schimba ulterior! Spre exemplu, daca folosim *SELECT * FROM t* pentru a defini un view, si ulterior definirii adaugam o noua coloana in tabela t, view-ul nu va returna automat si noua coloana!

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

7.1.2.5. Restrictii la definirea unui view

La crearea unui view trebuie tinut cont de urmatoarele restrictii:

- nu poate fi creat un view temporar (folosind clauza TEMPORARY, acceptabila in cazul unei tabele)
- un view nu poate avea asociate triggere (vezi lectia despre triggere)
- instructiunea SELECT ce defineste view-ul:
  - trebuie sa refere numai tabele si view-uri deja existente
  - nu poate referi tabele temporare
  - nu poate contine o subinterogare in FROM (in locul unei tabele)
  - nu poate referi variabile de orice natura (de sistem, locale, parametri sau variabile locale ale unei rutine, parametri de prepared statement etc)
  - poate contine ORDER BY, insa acesta va fi ignorat daca view-ul este folosit in cadrul unei instructiuni care are la randul sau ORDER BY

7.1.3. View-uri actualizabile

7.1.3.1. Conditii si tipuri de actualizare

Un view actualizabil este unul caruia i se pot aplica instructiuni DML de modificare a datelor (INSERT, UPDATE, DELETE). Restrictiile ce se aplica unui view pentru ca el sa poata fi actualizat difera in functie de instructiunea DML ce dorim sa functioneze:

- un view poate suporta instructiunea UPDATE ("**updatable**"). Pentru ca un view sa fie updatable avem doua conditii de baza:
  - corespondenta 1 la 1 intre liniile view-ului si ale tabelei din care provine informatia. Aceasta se poate obtine in primul rand respectand restrictiile pentru folosirea algoritmului MERGE
  - coloanele ce se doreste a fi modificate trebuie sa provina direct din coloane ale tabelei, nu sa fie calculate pe baza de expresii

```
-- View  neactualizabil deoarece coloanele sunt obtinute pe baza de expresii
CREATE VIEW v1 AS SELECT Pret*1.24 AS PretCuTVA, 4, CONCAT('produs: ',NumeProdus)

-- View  neactualizabil deoarece este obtinut prin functii de agregare:
--     unei linii din view ii pot corespunde mai multe linii din tabela de origine
CREATE VIEW v2 AS SELECT P_Categ,COUNT(*) FROM Produse GROUP BY P_Categ
```

Atunci cand un view respecta prima conditie dar numai o parte dintre coloane o respecta pe a doua, doar acele coloane vor putea participa in instructiuni UPDATE aplicate view-ului:

```
-- View  partial actualizabil
CREATE VIEW v3 AS SELECT NumeProdus, Pret*1.24 AS PretCuTVA

UPDATE v3 SET NumeProdus = 'Mere' WHERE NumeProdus = 'mere'
Query OK, 2 rows affected (0.00 sec)

UPDATE v3 SET PretCuTVA = 5 WHERE NumeProdus = 'Mere'
ERROR 1348 (HY000): Column 'PretCuTVA' is not updatable
```

- un view poate suporta instructiunea INSERT ("**insertable**"). In acest scop el trebuie sa fie un view updatable si sa respecte suplimentar cateva restrictii:
  - view-ul trebuie sa contina toate coloanele din tabela de baza care nu au valoare default
  - toate coloanele din view trebuie sa provina direct din coloane ale tabelei (sa nu fie calculate pe baza de expresii)

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

Nota: *astfel, un view care contine atat coloane provenite direct din tabela de baza cat si coloane calculate pe baza de expresii nu va fi insertable, dar poate fi updatable pe portiuni.*

7.1.3.2. Mentinerea consistentei datelor

In cazul view-urilor actualizabile, exista posibilitatea introducerii in tabela de baza, prin intermediul view-ului, a unor valori care nu ar fi vizibile in view. Sa consideram cazul unei tabele *Produse* cu coloanele *Nume* si *Pret* cu urmatorul continut:

```
SELECT * FROM Produse;
+----------+------+
| Denumire | Pret |
+----------+------+
| laptop   | 2000 |
| bocanci  | 1100 |
+----------+------+
```

Fie urmatorul view creat pe baza tabelei Produse:

```
CREATE VIEW ProduseScumpe SELECT * FROM Produse WHERE Pret>1000
```

View-ul este updatable si insertable - are corespondenta a liniilor 1 la 1 cu tabela de baza si contine numai coloane “curate”. Prin intermediul acestui view am putea incerca sa introducem in tabela de baza un nou produs, avand insa pretul mai mic decat 1000:

```
INSERT INTO ProduseScumpe VALUES('mere',5);
Query OK, 1 row affected (0.00 sec)
```

Observam ca, desi adaugarea inregistrarii s-a efectuat cu succes, ea nu este selectata de catre view (desi a fost introdusa prin intermediul lui!):

```
SELECT * FROM Produse
+----------+------+
| Denumire | Pret |
+----------+------+
| laptop   | 2000 |
| bocanci  | 1100 |
| mere     |    5 |
+----------+------+

SELECT * FROM ProduseScumpe
+----------+------+
| Denumire | Pret |
+----------+------+
| laptop   | 2000 |
| bocanci  | 1100 |
+----------+------+
```

In scopul interzicerii unor astfel de operatii exista clauza **WITH CHECK OPTION**, aplicabila numai view-urilor actualizabile. In prezenta ei, MySQL nu va permite decat modificari ale datelor pentru care clauza WHERE ramane true. In exemplul anterior, noua inregistrare introdusa nu respecta clauza WHERE; de asemenea, daca am fi incercat un UPDATE care modifica pretul laptop-ului la 990, aceasta ar fi creat o situatie asemanatoare, clauza WHERE nefiind respectata. Ambele operatii ar fi respinse in cazul unui view care foloseste in definitia sa WITH CHECK OPTION.

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

Clauza WITH CHECK OPTION are doua moduri de functionare:

- WITH LOCAL CHECK OPTION va verifica numai clauza WHERE a view-ului pentru care se aplica operatia
- WITH CASCADED CHECK OPTION se va asigura atat de respectarea clauzei WHERE a view-ului pentru care se aplica operatia, cat si a eventualelor view-uri pe baza carora acesta a fost creat (verificare resucrsiva)

Atunci cand nu se specifica tipul de verificare dorit, valoarea implicita este WITH CASCADED CHECK OPTION.

Atentie! Clauza WITH CHECK OPTION este permisa numai in cazul view-urilor actualizabile; daca view-ul nu este actualizabil, MySQL va produce o eroare la crearea view-ului. Aceasta inseamna, spre exemplu, ca WITH CHECK OPTION nu poate fi folosit impreuna cu ALGORITHM=TEMPTABLE.

7.1.4. Modificarea unui view

Singura modificare permisa asupra unui view este schimbarea definitiei acestuia. Sintaxa este foarte asemanatoare cu cea de creare, cu doua exceptii: se foloseste instructiunea ALTER si nu poate fi folosita clauza OR REPLACE:

```
ALTER VIEW ProduseScumpe AS SELECT * FROM Produse WHERE Pret>500
```

7.1.5. Stergerea unui view

Stergerea view-urilor se realizeaza cu instructiunea DROP VIEW care poate fi urmata de unul sau mai multe nume de view-uri; in ultimul caz numele trebuie separate prin virgula:

```
DROP VIEW [IF EXISTS] v1, v2, .....
```

Clauza IF EXISTS transforma eventuala eroare cauzata de absenta unui view in warning, vizualizabil cu SHOW WARNINGS.

Atunci cand instructiunea primeste ca argumente mai multe view-uri, dintre care doar o parte exista, vor fi sterse cele existente si se va genera o eroare legata de cele absente:

```
CREATE VIEW v1 AS SELECT 1;
DROP VIEW v1,v2,v3;
ERROR 1051 (42S02): Unknown table 'v2,v3'
SHOW TABLES LIKE 'v';
Empty set (0.00 sec)
```

7.1.6. Vizualizarea detaliilor unui view

Odata creat un view, detaliile sale pot fi vizualizate folosind instructiuni cunoscute de la lucrul cu tabele:

- **DESCRIBE numeview** va arata caracteristicile coloanelor componente ale view-ului
- **SHOW CREATE TABLE numeview** va arata instructiunea SQL ce defineste view-ul. Exista si sinonimul SHOW CREATE VIEW

InfoAcademy

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

```
CREATE VIEW v AS SELECT 2;
DESCRIBE v;
+-------+--------+------+-----+---------+-------+
| Field | Type   | Null | Key | Default | Extra |
+-------+--------+------+-----+---------+-------+
| 2     | int(1) | NO   |     | 0       |       |
+-------+--------+------+-----+---------+-------+

SHOW CREATE TABLE v; -- functioneaza si SHOW CREATE VIEW
*************************** 1. row ***************************
       View: v
Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost`
                    SQL SECURITY DEFINER VIEW `v` AS select 2 AS `2`
```

7.1.7. Vizualizarea listei de view-uri

View-urile sunt si ele prezente in rezultatul instructiunii SHOW TABLES. Pentru a distinge intre view-uri si tabele avem la dispozitie clauza suplimentara FULL, care produce o coloana aditionala cu tipul tabelei:

```
SHOW FULL TABLES;
+------------------------+------------+
| Tables_in_test         | Table_type |
+------------------------+------------+
| Produse                | BASE TABLE |
| ProduseScumpe          | VIEW       |
....

-- pentru a obtine doar view-urile putem filtra lista dupa valoarea celei de-a doua coloane:
SHOW FULL TABLES WHERE Table_type='view';
```

Nota: *o alta modalitate – indirecta – de a determina daca o tabela este un view sau tabela reala este folosirea instructiunii SHOW TABLE STATUS.*

7.1.8. Verificarea unui view

La crearea unui view, MySQL se asigura de existenta tuturor tabelelor si view-urilor pe baza carora acesta este construit. Ulterior insa acestea se pot modifica sau disparea, ceea ce va invalida view-ul creat. Putem verifica validitatea unui view folosind instructiunea CHECK TABLE:

```
CREATE TABLE t(i int);
CREATE VIEW v AS SELECT * FROM t;
DROP TABLE t;
CHECK TABLE v\G
*************************** 1. row ***************************
   Table: test.v
      Op: check
Msg_type: Error
Msg_text: Table 'test.t' doesn't exist
*************************** 2. row ***************************
   Table: test.v
      Op: check
Msg_type: Error
Msg_text: View 'test.v' references invalid table(s) or column(s) or function(s) or
definer/invoker of view lack rights to use them
*************************** 3. row ***************************
   Table: test.v
      Op: check
Msg_type: error
Msg_text: Corrupt
```

InfoAcademy

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

7.2. Variabile definite de client (“user variables”)

7.2.1. Concepte si utilitate

O variabila MySQL reprezinta un nume atribuit unor date, permitandu-ne sa memoram valori in scopul folosirii lor ulterioare. Iata cateva exemple de utilizare:

- in cadrul unui script SQL, valorile care apar in mod repetat si care au sanse sa se modifice pe viitor pot fi definite la inceputul scriptului sub forma de variabile (ceea ce le face si mai vizibile)
- rezultatul unei operatii SQL poate fi salvat intr-o variabila si folosit ulterior de oricate ori, fara a repeta operatia de fiecare data

7.2.2. Numele unei variabile

O variabila definita de client are notatia **@numevariabila**, unde numele poate contine caractere alfanumerice, punct, underscore si dolar. Numele de variabila poate contine si alte caractere daca este incardat intre delimitatorii de string (ex: @'numar-produse').

Numele de variabile sunt case insensitive incepand cu MySQL 5.0 si case sensitive in versiunile anterioare.

7.2.3. Tipul de date al unei variabile

Tipul de date al unei variabile poate fi numar intreg, numar fractionar, sir de caractere sau NULL. Tipul de date al variabilei nu este impus de la bun inceput (variabilele nu se declara), ci se poate schimba de-a lungul existentei variabilei in functie de natura valorilor atribuite acesteia. Valoarea atribuita poate fi una constanta sau rezultatul unei expresii sau interogari, in ambele cazuri ea avand un tip de date bine stabilit:

- daca este vorba de o valoare constanta specificata in cadrul instructiunii SQL (un asa-numit “literal”), tipul de date va depinde de felul in care este reprezentata valoarea: numerele care nu contin separator zecimal sunt automat considerate intregi, numerele cu separator zecimal fractionare, iar includerea valorii intre apostroafe cauzeaza tratarea ei ca sir de caractere
- valorile produse de expresii au un tip de date determinat la evaluarea expresiei

Nota: *o variabila careia i-a fost atribuita o valoare de tip sir de caractere va avea acelasi character set si collation ca string-ul atribuit.*

7.2.4. Definirea unei variabile

Variabilele pot fi definite in doua moduri:

- folosind instructiunea SET, care permite definirea uneia sau mai multor variabile simultan
- in cadrul unei expresii, folosind operatorul de atribuire

Sintaxa instructiunii SET este:

```
SET @numevariabila1 = expresie1 [, @numevariabila2 = expresie2] ...
```

Ca operator de atribuire poate fi folosit = sau := (cel in urma fiind, de altfel, singurul recomandat atunci cand o variabila primeste o valoare in cadrul unei instructiuni non-SET).

Exemple:

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

```
SET
      @nr=1,
      @nr_aleator = FLOOR(RAND()*100),
      @maine := CURDATE() + INTERVAL 1 DAY,
      @nr1 = @nr+1;       -- @nr1 capata valoarea NULL deoarece @nr are pe moment valoarea NULL!
SET   @nr2 = @nr+1;       -- valoarea unei variabile poate fi obtinuta pe baza alteia
```

Dupa cum se observa in exemplul de mai sus, valoarea unei variabile poate fi o constanta sau rezultatul unei expresii, iar expresia poate contine inclusiv valori ale altor variabile definite anterior.

ATENTIE!

- **daca referim o variabila nedefinita anterior, ea are automat valoarea NULL! Acest lucru va face majoritatea expresiilor in care ea participa sa produca valoarea NULL.**
- **daca valoarea unei expresii e calculata pe baza unei variabile definite anterior in cadrul aceleiasi instructiuni SET, valoarea variabilei este NULL! (vezi exemplul de mai sus, cazul lui @nr1)**

O variabila poate primi valoare si in cadrul altei instructiuni decat SET, caz in care este necesara folosirea operatorului := (operatorul = este considerat automat a fi operatorul de comparare in aceste contexte):

```
-- Daca variabila n nu a fost definita anterior:
SELECT @n=10, @n:=10; /* in prima parte @n are valoarea NULL si este comparat cu 10, rezultand
                         NULL; abia in a doua parte primeste valoare */
+-------+--------+
| @n=10 | @n:=10 |
+-------+--------+
|  NULL |     10 |
+-------+--------+
```

7.2.5. Utilizarea si domeniul de vizibilitate al unei variabile

O variabila exista numai in cadrul si pe parcursul sesiunii in care a fost definita:

- variabilele definite de catre un client nu pot fi accesate de catre ceilalti
- toate variabilele create de catre un client dispar in momentul deconectarii acestuia de la serverul MySQL.

Variabilele definite de client sunt automat vizibile in interiorul codului memorat in baza de date (proceduri, functii, triggere, evenimente) fara a fi nevoie de masuri speciale. Mai mult, o astfel de variabila definita in cadrul unei proceduri sau functii continua sa fie disponibila si dupa incheierea executiei rutinei respective.

O variabila poate fi utilizata in orice context in care este vizibila,acolo unde este nevoie de o valoare – exemple:

- ca operand intr-o expresie
- ca argument al unei functii predefinite sau definite de catre client

Variabilele nu pot fi insa utilizate acolo unde sintaxa SQL cere o constanta (“literal”) - iata exemple:

- in cadrul clauzei LIMIT (ex: LIMIT @nr_randuri)
- in cadrul clauzei LOAD DATA INFILE pentru a specifica calea catre fisierul de importat (ex: LOAD DATA INFILE @cale_fisier ....)

7.2.6. Alte tipuri de variabile

MySQL dispune si de alte cateva tipuri de variabile, care **nu trebuie confundate cu variabilele discutate mai sus**:

InfoAcademy

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

- variabile locale, declarate in cadrul functiilor sau procedurilor memorate in baza de date. Aceste variabile au domeniu de vizibilitate mult mai restrans (doar functia/procedura in care sunt definite) si se utilizeaza cu simplul lor nume, fara a fi precedate de @
- parametri ai functiilor sau procedurilor definite de client – sunt un caz special de variabile locale, aplicandu-li-se aceleasi reguli ca mai sus
- variabile de sistem – reprezinta setari ale serverului, unele dintre ele putand fi modificate de catre client (ex: sql_mode)
- variabile de stare – sunt definite de catre MySQL si memoreaza diferite informatii si statistici legate de functionarea serverului, utile administratorului de server pentru a mentine si optimiza serverul de baze de date

7.3. Prepared statements (interogari precompilate)

7.3.1. Ce este un prepared statement si avantajele sale

Un prepared statement este o interogare precompilata creata de catre un client in scopul executarii sale ulterioare cu diferite seturi de date de intrare. Sa luam exemplul unei tabele SQL folosita pentru logging, in care un site memoreaza diferitele evenimente legate de functionarea sa, fiecare eveniment fiind consemnat printr-o interogare de forma urmatoare:

```
INSERT INTO loguri(moment, text) VALUES('2010-11-19','Userul andrei conectat');
```

La fiecare executie a acestei interogari de catre codul site-ului web (lucru care se va intampla destul de des) au loc urmatoarele operatii:

- interogarea este pregatita de catre codul client si trimisa serverului SQL. Daca acesta nu se afla pe aceeasi masina cu serverul web, se efectueaza trafic in retea
- interogarea este receptionata de catre serverul SQL si parsata (instructiunea SQL este validata, identificandu-se in cadrul ei diferitele elemente de sintaxa)
- serverul creeaza o forma interna, executabila, a instructiunii, pe care apoi o proceseaza

Observam ca primii 2 pasi vor fi identici de fiecare data, singurul lucru care difera de la o executie la alta fiind setul de date de intrare (momentul de timp si textul asociat intrarii din log).

Un prepared statement permite executarea o singura data a pasilor initiali:

- clientul creeaza un prepared statement, trimitand serverului instructiunea SQL dorita si specificand locurile din cadrul ei in care vor fi introduse valorile ce parametrizeaza interogarea
- serverul precompileaza interogarea si asteapta executarea ei viitoare la solicitarea clientului
- clientul apeleaza interogarea precompilata ori de cate ori doreste, specificand la fiecare apelare doar setul de parametri dorit

In acest fel se minimizeaza traficul prin retea si operatiile efectuate de catre server.

Accesul la facilitatea de prepared statements a unui server MySQL este posibil in doua moduri:

- prin intermediul API-ului bibliotecilor puse la dispozitie de catre serverul MySQL. Interogarile precompilate sunt folosite preponderent de catre aplicatii, nu de catre utilizatorul uman in mod interactiv
- prin intermediul catorva instructiuni SQL, ceea ce permite testarea in mod interactiv a interogarilor precompilate inainte ca acestea sa fie folosite in cadrul unei aplicatii. Instructiunile folosite sunt:
  - PREPARE – pentru crearea unei interogari precompilate
  - EXECUTE – pentru executarea cu un anumit set de parametri a unei interogari precompilate creata anterior cu PREPARE

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

- DELLOCATE PREPARE – pentru stergerea unei interogari precompilate de care nu mai este nevoie

Materialul de fata va aborda, desigur, cazul utilizarii interactive.

7.3.2. Crearea unui prepared statement

Pentru crearea unei interogari precompilate in modul interactiv se foloseste instructiunea PREPARE, specificand query-ul dorit si inlocuind parametrii ce vor fi pasati la executie cu semne de intrebare:

```
PREPARE ps FROM 'SELECT * FROM OAMENI WHERE Inaltime>? AND Nume = ?'
```

Atentie! Chiar daca parametrii pasati la executie vor fi de tip sir de caractere, ei nu trebuie incadrati intre apostroafe in definitia prepared statement-ului!

Textul interogarii poate fi specificat si sub forma unei variabile:

```
@query = 'SELECT * FROM OAMENI WHERE Inaltime>? AND Nume = ?';
PREPARE ps FROM @query
```

Daca deja exista un prepared statement cu acelasi nume, serverul il va dezaloca automat pe cel vechi inainte de a-l crea pe cel nou; daca crearea celui nou esueaza, cel vechi ramane sters!

Un prepared statement poate contine o singura interogare componenta iar in cadrul acesteia este permis numai un subset al instructiunilor SQL:

- SELECT, INSERT, UPDATE, DELETE, REPLACE
- CREATE TABLE
- SET, DO si diverse SHOW-uri

Un prepared statement nu poate folosi in cadrul sau instructiunile PREPARE, EXECUTE sau DEALLOCATE PREPARE.

Odata creat, un prepared statement este disponibil in cadrul si pe parcursul sesiunii in care a fost creat:

- este vizibil doar in cadrul acelei sesiuni de client, nu si in altele
- chiar daca este definit in interiorul unui bloc sau al unei functii/proceduri, el este vizibil in intreaga sesiune, chiar si dupa ce executia functiei/procedurii s-a incheiat
- este dezalocat automat la inchiderea sesiunii clientului

7.3.3. Executarea unui prepared statement

A executa un prepared statement inseamna a apela o interogare precompilata de pe server, specificand setul de parametri de intrare ai acesteia. Folosim in acest scop instructiunea EXECUTE impreuna cu numele interogarii precompilate, pasand datele de intrare sub forma de variabile, dupa cum urmeaza:

```
CREATE TABLE Oameni(inaltime INT, nume VARCHAR(200));
INSERT INTO Oameni VALUES(180, 'Mihai');
PREPARE ps FROM 'SELECT * FROM Oameni WHERE inaltime>? AND nume LIKE ?';
Statement prepared

SET @h=170,@n='%a%';
EXECUTE ps USING @h,@n;
```

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

```
+----------+-------+
| inaltime | nume  |
+----------+-------+
|      180 | Mihai |
+----------+-------+
```

La executie, clientul trebuie sa furnizeze valori pentru toti parametrii specificati sub forma de ? la definirea prepared statement-ului, in caz contrar MySQL semnalizand eroarea:

```
EXECUTE ps USING @h;
ERROR 1210 (HY000): Incorrect arguments to EXECUTE
```

7.3.4. Stergerea unui prepared statement

Interogarile precompilate create de client pot fi dezalocate in doua moduri:

- automat, la inchiderea sesiunii clientului
- manual, folosind instructiunea DEALLOCATE PREPARE sau DROP PREPARE

```
-- urmatoarele doua intructiuni sunt echivalente
DEALLOCATE PREPARE ps
DROP PREPARE ps
```

7.4. Tranzactii

7.4.1. Conceptul de tranzactie

O tranzactie reprezinta un set de instructiuni SQL care se executa in mod atomic:

- daca vreuna dintre instructiuni esueaza, se poate readuce baza de date in starea de dinaintea inceperii tranzactiei (“ROLLBACK”)
  - efectul tuturor instructiunilor precedente este anulat
  - toate instructiunile ce urmeaza celei “vinovate” nu se mai executa
- daca toate instructiunile componente se incheie cu succes, abia atunci sunt efectuate modificarile permanente in baza de date (“COMMIT”)
- pe parcursul tranzactiei, numai clientul care a initiat-o vede schimbarile produse

Scopul tranzactiilor este de a ne asigura ca operatiile care implica mai multe interogari SQL nu lasa modificari partiale in caz de eroare. Spre exemplu, in domeniul bancar, la transferul banilor dintr-un cont in altul distingem doua operatii distincte: scaderea balantei contului sursa si cresterea balantei contului destinatie. Daca, dintr-un motiv oarecare, cea de-a doua operatie esueaza si nu poate fi efectuata, este necesara “punerea la loc” a banilor in contul sursa, in caz contrar titularul contului sursa fiind vaduvit de suma implicata in incercarea de transfer. Dimpotriva, daca prima operatie esueaza (contul sursa nu este debitat), cea de-a doua nu mai trebuie efectuata.

Tranzactiile mai sunt de asemenea utilizate pentru izolarea clientilor unul fata de celalalt in cazul accesului concurent la informatie: cat timp o tranzactie este in desfasurare, modificarile efectuate in cadrul ei nu sunt vizibile decat pentru clientul care o efectueaza.

Nota: *MySQL suporta tranzactii numai pentru tabelele cu anumite storage engine-uri, cel mai important fiind InnoDB.*

InfoAcademy

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

7.4.2. Tranzactiile si modul autocommit

Serverul MySQL functioneaza implicit in mod *autocommit* – efectul fiecarei instructiuni este memorat permanent in baza de date imediat dupa executarea instructiunii, devenind astfel vizibil si pentru ceilalti clienti. Constatam ca acesta este un caz foarte particular de tranzactie – una care contine o singura instructiune. A crea o tranzactie ce contine mai multe instructiuni inseamna a dezactiva modul autocommit si a face manual commit-ul in momentul incheierii cu succes a tuturor instructiunilor componente ale tranzactiei.

Modul autocommit poate fi dezactivat in doua feluri:

- prin setarea variabilei *autocommit* pe 0 (efect persistent). MySQL nu va produce modificarile permanente in baza de date decat atunci cand clientul executa instructiunea COMMIT. Dupa executarea ei incepe automat o noua tranzactie, deoarece autocommit ramane pe valoarea 0

```
SET autocommit = 0;
```

- prin folosirea instructiunilor de demarare a unei tranzactii, START TRANSACTION sau BEGIN (efect temporar, sau mai bine zis dependent de valoarea curenta a lui autocommit). Diferenta este ca START TRANSACTION suspenda temporar modul autocommit, urmand ca, dupa executarea COMMIT-ului sau ROLLBACK-ului tranzactiei, vechiul mod autocommit sa fie reluat:
  - daca initial autocommit-ul era pus pe 1, el devine 0 pe durata tranzactiei si revine la 1 dupa COMMIT/ROLLBACK
  - daca initial era pus pe 0, atunci, dupa COMMIT/ROLLBACK, incepe implicit o noua tranzactie

Nota: *daca in cadrul tranzactiei sunt folosite tabele al caror storage engine nu suporta tranzactii, ele vor functiona in mod autocommit, emitandu-se warning-uri in cazul incercarii de ROLLBACK.*

Exemplu:

```
-- O tabela tranzactionala
CREATE TABLE Produse (Nume VARCHAR(200), Pret INT) ENGINE InnoDB;
START TRANSACTION;
INSERT INTO Produse VALUES('mere',5), ('pere',6);
SELECT * FROM Produse; -- in timpul tranzactiei clientul vede efectele ei
+------+------+
| Nume | Pret |
+------+------+
| mere |    5 |
| pere |    6 |
+------+------+

ROLLBACK;
SELECT * FROM Produse; -- modificarile au fost anulate
Empty set (0.00 sec)

-- O tabela non-tranzactionala
CREATE TABLE test (i INT) ENGINE MyISAM;
BEGIN;
INSERT INTO test VALUES(10);
ROLLBACK;
Query OK, 0 rows affected, 1 warning (0.00 sec)
SHOW WARNINGS;
+---------+------+----------------------------------------------------------------+
| Level   | Code | Message                                                        |
+---------+------+----------------------------------------------------------------+
| Warning | 1196 | Some non-transactional changed tables couldn't be rolled back  |
+---------+------+----------------------------------------------------------------+
1 row in set (0.00 sec)
```

InfoAcademy

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

7.4.3. Incheierea unei tranzactii

Cele doua moduri posibile de incheiere a unei tranzactii sunt:

- **COMMIT** - modificarile produse de instructiunile componente ale tranzactiei devin permanente, fiind salvate in baza de date
- **ROLLBACK** - modificarile produse de instructiunile componente ale tranzactiei executate pana atunci sunt anulate

Atentie! Nu se poate face ROLLBACK pentru instructiunile DDL, ci numai pentru cele DML, si numai daca tabela asupra careia se produc modificarile suporta tranzactii!

O tranzactie poate lua sfarsit ca urmare a diferite evenimente, unele dintre ele producand un COMMIT al tranzactiei curente, altele un ROLLBACK:

- evenimente care cauzeaza COMMIT:
  - clientul executa explicit instructiunea COMMIT. Dupa executie, modul autocommit revine la valoarea de dinaintea inceperii tranzactiei
  - clientul demareaza o noua tranzactie. Ca urmare, tranzactia in desfasurare va fi incheiata automat cu COMMIT
  - clientul executa o instructiune DDL. Ca urmare, tranzactia in desfasurare va fi incheiata automat cu COMMIT
- evenimente care cauzeaza ROLLBACK:
  - clientul executa instructiunea ROLLBACK. Dupa executie, modul autocommit revine la valoarea de dinaintea inceperii tranzactiei
  - clientul se deconecteaza. Tranzactia in desfasurare va suferi ROLLBACK

7.5. BIBLIOGRAFIE

- MySQL Reference Manual
  - View-uri:
    - http://dev.mysql.com/doc/refman/5.1/en/views.html
    - http://dev.mysql.com/doc/refman/5.1/en/create-view.html
    - http://dev.mysql.com/doc/refman/5.1/en/faqs-views.html
    - http://dev.mysql.com/doc/refman/5.1/en/view-restrictions.html
  - Variabile
    - http://dev.mysql.com/doc/refman/5.1/en/user-variables.html
  - Prepared statements
    - http://dev.mysql.com/doc/refman/5.1/en/sql-syntax-prepared-statements.html
  - Tranzactii
    - http://dev.mysql.com/doc/refman/5.1/en/sql-syntax-transactions.html
    - http://dev.mysql.com/doc/refman/5.1/en/cannot-roll-back.html
    - http://dev.mysql.com/doc/refman/5.1/en/implicit-commit.html
- Carti:
  - MySQL 5 Certification Study Guide – cap. 14, 16, 17
  - MySQL (4th Edition) – cap. 2.11, 2.13

InfoAcademy

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.