Laborator 5

Vederi

Vederile sunt tabele virtuale care permit programatorului să creeze imagini ale datelor. Spre deosebire de tabelele care conţin date, vederile nu conţin decât interogări care regăsesc în mod dinamic datele atunci când sunt utilizate.

Vederile se pot utiliza pentru reutilizarea instrucţiunilor SQL şi pentru reutilizarea interogărilor, astfel ajungându-se la simplificarea operaţiunilor SQL simple. De asemenea, se poate modifica formatarea şi reprezentarea tabelelor cu ajutorul vederilor.

Ca şi relaţiile de bază, şi vederile pot fi actualizate ceea ce atrage modificarea tabelelor statice din care derivă. Datele unei vederi nu sunt memorate într-un obiect al bazei de date, ceea ce se memorează este o frază SELECT pe baza căreia datele sunt determinate în momentul invocării vederii, precum o interogare prememorată. O vedere se poate creea numai dacă nu a fost creată anterior [4],[9].

- **Instrucţiunea CREATE VIEW**

```
CREATE
[OR REPLACE]
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { utilizator | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW nume_vedere [(lista de coloane)]
AS clauza select
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

Unde

- **OR REPLACE**- este folosit dacă se doreşte înlocuirea unei vederi cu o altă vedere
- **ALGORITHM** – clauza care determină cum MySQL procesează vederea
- **DEFINER, SQL SECURITY**- clauze care specifică elemente de securitate
- **WITH CHECK OPTION** –clauza pentru a constrânge inserarea sau actualizarea rândurilor din tabelă referitoare la vedere. Aceasta forţează toate operaţiile de modificare a datelor, referitoare la vedere să adere la criteriile stabilite de instrucţiunea SELECT. Dacă are loc modificarea unei tuple din vedere, clauza aceasta garantează faptul că datele modificate rămân vizibile în vedere.
- **Nume_vedere** – reprezintă numele vederii care ca fi creată

- **Lista_coloane**-reprezintă coloanele care vor fi incluse în vederea respectivă
- **Clauza select**- va selecta din tabelele din care se doreşte crearea vederii ceea ce se doreşte a fi integrat

Pentru regăsirea datelor care fac parte din vederea creată se va folosi o clauză de tip SELECT.[9]

Exemplul 1:

Un exemplu de creare a unei vederi cu numele test dintr-o tabelă angajaţi, care să conţină toate datele din tabela angajaţi este următorul:

```
CREATE VIEW test AS SELECT * FROM angajati;
```

Pentru a vizualiza datele din vederea test, se va scrie următoarea instrucţiunea:

```
SELECT * FROM test;
```

- **Instrucţiunea ALTER VIEW**

Modificarea unei vederi create anterior se poate face cu instrucţiunea ALTER VIEW.[9]

```
ALTER
[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
[DEFINER = { utilizator | CURRENT_USER }]
[SQL SECURITY { DEFINER | INVOKER }]
VIEW nume_vedere [(lista_coloane)]
AS clauza_select
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

- **Instrucţiunea DROP VIEW**

Această instrucţiune şterge una sau mai multe vederi din baza de date.

```
DROP VIEW {nume_vedere}[,n]
```

Exemplul 2:

Un exemplu ar fi ştergerea vederii create anterior prin instrucţiunea:[9]

```
DROP VIEW test;
```

Utilizarea indecşilor

Un index este un obiect al schemei bazei de date care facilitează accesarea rapidă şi univocă a datelor. În anumite condiţii viteza de execuţie a cererilor. După ce indexul a fost creat, utilizatorului nu i se mai cere nici o operaţie directă asupra lui, el va fi folosit şi întreţinut automat de către SGBD.

Pentru tabele mici, folosirea indecşilor nu aduce îmbunătăţiri de performanţă. Se folosesc atunci când coloanele după care se creează indecşii conţin o diversitate mare de informaţii sau multe valori NULL. Indecşii optimizează interogările atunci când acestea returnează o cantitate mică de date. Indecşii cresc viteza de regăsire a datelor dar încetinesc actualizarea datelor datorită faptului că sistemul trebuie să actualizeze şi fişierele index. În general este foarte util să se creeze indecşi după câmpuri care se folosesc în operaţii de join. SQL Server foloseşte indecşi pentru optimizarea interogărilor pe care le are de rezolvat atunci când construieşte planul de execuţie al fiecărei interogări. Deşi acest proces nu poate fi urmărit de către utilizator,acesta va putea participa prin definirea anterioară a coloanelor care vor fi indexate. De regulă se indexează coloanele care figurează mai frecvent ca şi criteriu de căutare în interogări.

Deoarece definirea indecşilor va avea şi dezavantaje precum consumul spaţiului din memorie şi încetinirea operaţiilor aplicate asupra tabelelor, se va dori evitarea excesului de indecşi. De asemenea utilizatorii nu pot vedea indecşii, dar vor putea observa viteza crescută a soluţionării interogărilor.

Definirea unor constrângeri de tip PRIMARY KEY şi UNIQUE se materializează, în mod automat, prin crearea de către server a indecşilor de cheie primară şi unici corespunzători.[4],[15]

- **Instrucţiunea CREATE INDEX**

Sintaxa pentru crearea unui index este următoarea:

```
CREATE INDEX nume_index ON nume_tabela (nume_coloana1 [,nume coloana2, …])
```

Exemplul 3:

Crearea unui index după coloana număr factură în tabela facturi:

```
CREATE INDEX nr_factura_index ON facturi (nrfact)
```

- **Instrucţiunea DROP INDEX**

Ştergerea unui index se face cu comanda DROP INDEX:

```
DROP INDEX nume_index  [ON nume_tabela]
```

- **Instrucţiunea SHOW INDEX**

Această comandă se poate folosi pentru a obţine o listă cu toţi indecşii asociaţi unei tabele.

```
SHOW INDEX FROM nume_tabel
```

Proceduri stocate

Una dintre opțiunile pentru stocarea și executarea programelor este folosirea procedurilor stocate. Aceste proceduri sunt similare cu cele din alte limbaje de programare, având în componență parametri de intrare și ieșire și instrucțiuni. De asemenea, din interiorul acestora se pot apela alte proceduri, iar programatorul poate seta o valoare de eroare ce va fi returnată în cazul în care nu se îndeplinesc anumite condiții.

Este de reținut că procedurile nu returnează o valoare, deci nu pot fi folosite direct în expresii.[9]

Procedurile stocate sunt colecții formate dintr-una sau mai multe instrucțiuni, salvate în vederea unei utilizări ulterioare. [4]

Dintre avantajele folosirii procedurilor stocate putem aminti:

- simplificarea unor operații complexe
- asigurarea consecvenței datelor, fapt care reiese din înlocuirea unei serii de etape cu o simplă apelare a unei proceduri stocate
- simplificarea gestiunii schimbărilor deoarece utilizatorul nu va mai fi nevoit să modifice într-un cod care conține foarte multe instrucțiuni, ci va modifica codul unei proceduri cu un scop precis
- întrucât procedurile stocate sunt de obicei stocate într-o formă compilată, programul SGBD va procesa mai rapid comanda, de unde rezultă și o îmbunătățire a performanțelor

Utilizarea procedurilor stocate are de asemenea și dezavantaje de care trebuie ținut cont, dintre care amintim faptul că au o sintaxă mai complexă decât a instrucțiunilor SQL simple, ceea ce necesită un nivel de experiență mai ridicat. De aici a pornit și limitarea impusă de anumiți administratori de baze de date asupra procedurilor stocate. [4] De asemenea ca dezavantaje pot apărea congestionarea serverului sau scăderea performanțelor acestuia.

- **Crearea și executarea procedurilor stocate**

Crearea procedurilor se face cu clauza CREATE PROCEDURE care are sintaxa:

```
CREATE
[DEFINER = { user | CURRENT_USER }]
PROCEDURE  nume_procedura ([_parametrii_proc[,...]])
[caracteristici ...] rutina
```

Parametrii_proc:
[IN | OUT | INOUT] param_name type

caracteristici:

```
COMMENT 'string'
| LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
```

În MySQL Workbench o procedură se creează astfel:

- Se va da click dreapta pe Stored Procedure şi se alege opţiunea Create Stored Procedure
- Se va deschide o fereastră unde se va poate scrie procedura

Fig.1.Deschiderea ferestrei în care se va scrie procedura

Exemplu:

- Vom dori ca un exemplu să creăm o procedură simplă care să afişeze atunci când e apelată conţinutul tabelei funcţii, deci codul nostru va arăta după cum este prezentat în figura 10.2, după care se apasă Apply
- Va apărea o fereastră care conţine codul final al procedurii stocate după care se apasă butonul Apply şi Finish;

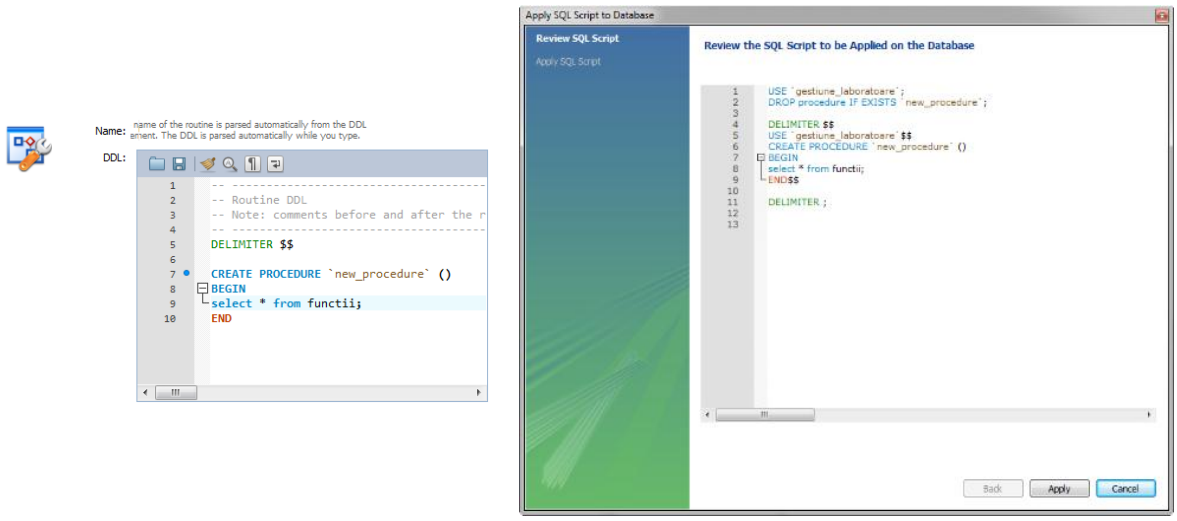

Fig. 2.Definitivarea procedurii

- Pentru a apela procedura creată se va scrie codul CALL new_procedure(); care va duce la afişarea tabelei funcţii [22]

- **Modificarea procedurilor stocate**

O procedură se poate modifica cu ajutorul clauzei ALTER PROCEDURE pentru ca utilizatorul să nu fie nevoit să şteargă procedura şi să o recreeze mai apoi. Un alt plus în cazul folosirii acestei proceduri este faptul că în acest caz procedurile sau triggerele dependente nu mai trebuie recompilate. [9]

Sintaxa pentru această clauză este următoarea:

ALTER PROCEDURE nume_procedura [caracteristici ...]

În MySQL Workbench se va da click dreapta pe numele procedurii care se doreşte a fi modificată şi se alege opţiunea alter procedure pentru a modifica instrucţiunile conţinute în acea procedură.

- **Stergerea procedurilor stocate**

Ştergerea unei proceduri se face cu ajutorul sintaxei

DROP PROCEDURE [IF EXISTS] nume_procedura

Triggere

Triggerele reprezintă o clasă specială de proceduri stocate, asociate unei tabele, definite pentru a fi lansate în execuţie automat la iniţierea unei operaţii de tip UPDATE, INSERT sau DELETE asupra tabelei în cauză.

Triggerele vor fi asociate cu o tabelă permanentă, deci nu pot fi create pentru o vedere sau o tabelă temporară.

Exemple tipice de utilizare a triggerelor sunt: cascadarea operațiilor de modificare de la tabela curentă la alte tabele din baza de date sau anularea modificărilor care ar duce la violarea integrității referențiale.

Un trigger este inițiat ori de câte ori se încearcă operația de modificare asupra tabelei căreia îi este atașat. De asemenea triggerul și operația atașată lui sunt considerate ca fiind un tot unitar, ceea ce duce la anularea operației care a declanșat triggerul în momentul în care execuția triggerului eșuează. Se cunoaște faptul că pentru un anumit tabel pot exista mai multe triggere [14].

Este de reținut faptul că există pe lângă triggerele simple și alte tipuri de triggere și anume:

Triggere multiple-care se referă la faptul că SQL Server permite crearea de mai multe triggere, cu nume diferite, pentru același tip de operație și același tabel

Triggere recursive-SQL Server permite două moduri de apel recursiv al triggerelor şi anume: recursivitate indirectă care se face prin intermediul altui trigger şi recursivitate directă unde triggerul este reactivat datorită modificărilor pe care el însuşi le face în baza de date.

Triggere imbricate-SQL Server permite apeluri imbricate de triggere până la 32 de nivele de adâncime, limitare care garantează că un apel imbricat poate fi terminat prin intervenţia sistemului în momentul depăşirii nivelului de imbricare

Sintaxa pentru crearea triggerelor este:

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_nume timp_trigger eveniment_trigger
ON tbl_name FOR EACH ROW corp_trigger
```

Crearea triggerelor se poate face dacă utilizatorul are privilegii asupra tabelei asociate triggerului. Clauza DEFINER determină verificarea privilegiilor de acces ale utilizatorului în momentul activării triggerului.

Timp_trigger este timpul de acţiune al triggerului care determină momentul în care este activat acesta şi anume înainte sau după fiecare rând care trebuie modificat prin opţiunile BEFORE sau AFTER

Eveniment_trigger indică tipul de clauză care activează triggerul care poate fi una dintre următoarele: INSERT, UPDATE, DELETE, REPLACE.

Corp_trigger reprezintă codul care trebuie executat când este activat triggerul. În cazul în care se doreşte executarea mai multor instrucţiuni se va folosi **BEGIN ... END**. Pentru a ne referi la coloanele tabelului asupra căruia se aplică triggerul se vor folosi **OLD** şi **NEW**. **OLD.numecol** se referă la o coloană a tabelului de dinainte de aplicarea procedurilor update sau delete.

NEW.numecol se referă la o coloană a unui rând nou care va fi inserat sau la un rând existent după ce a fost updatat tabelul.[23]

Exemplu:

Astfel, vom creea un trigger care să schimbe numele unuia dintre responsabilii conţinuţi în tabelul responsabili. Pentru a urmări mai bine modul în care se proedează, vom creea un nou tabel care să conţină câmpul modificat şi două câmpuri suplimentare şi anume action care ne va arăta ce acţiune s-a efectuat asupra acestui câmp şi changedon care va conţine date despre momentul în care a fost efectuată acţiunea.

Tabelul va fi creat cu codul:

```
use gestiune_laboratoare;
CREATE TABLE if not exists responsabili_nou(ID INTEGER NOT NULL,
                                nume VARCHAR(20) NULL,
                                prenume VARCHAR(20) NULL,
                                adresa VARCHAR(100) NULL,
                                ID_functie INTEGER NOT NULL,
                                varsta int(20),
                                action varchar(50) default null,
                                changedon datetime DEFAULT NULL,
                                PRIMARY KEY(ID),
                                FOREIGN KEY(ID_functie) REFERENCES functii(ID));
```

Dacă vom afişa acest tabel, vom observa că toate câmpurile acestuia au valoarea NULL. Următorul pas va fi crearea triggerului care se numeşte **responsabili_update** şi care va modifica numele responsabilului Gog Grigore în Vana Grigore. Sintaxa pentru trigger este:

```
DELIMITER $$
CREATE TRIGGER responsabili_update
   before UPDATE ON responsabili
   FOR EACH ROW BEGIN
    INSERT INTO responsabili_nou
   SET action = 'update',
      nume = new.nume,
      prenume = OLD.prenume,
      adresa = OLD.adresa,
                  ID_functie=OLD.ID_functie,
                  varsta=OLD.varsta,
                  changedon=NOW();
END$$
```

La rularea acestui trigger, în navigator se poate observa că la secţiunea trigger a tabelului responsabili a apărut în arborele desfăşurat şi triggerul creat.

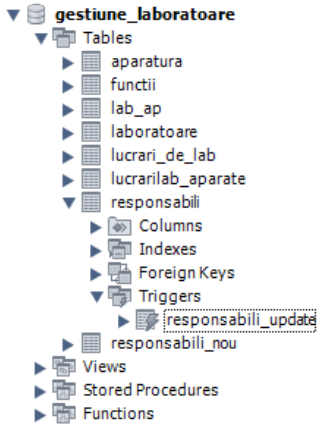

Fig. 3.Vizualizarea triggerului creat în arborele desfăşurat al bazei de date

Pentru a vedea dacă acesta funcţionează, se rulează instrucţiunea:

```
update responsabili
set nume='vana'
where prenume='grigore';
```

În momentul în care se rulează instrucţiunea:

```
select * from responsabili_nou;
```

se va afişa tabelul responsabili_nou rezultat în urma aplicării triggerului:

ID	nume	prenume	adresa	ID_functie	varsta	action	changedon
0	vana	Grigore	Str. Observatorului nr.5	2	45	update	2013-09-19 10:23:26
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fig. 4.Vizualizarea tabelului nou creat cu ajutorul triggerului.

Pentru a şterge un trigger se foloseşte sintaxa:

DROP TRIGGER {nume_trigger}[,n]

Exemplu:

Dacă se doreşte ştergerea triggerului creat anterior se va folosi instrucţiunea:

```
drop trigger responsabili_update;
```

Aspecte legate de securitate în MySQL

- **Instrucţiunea CREATE USER**

```
CREATE USER user_specification [, user_specification] ...
user_specification:
user [IDENTIFIED BY [PASSWORD] 'password'| IDENTIFIED WITH auth_plugin [AS
'auth_string']]
```

Clauza CREATE USER creează noi conturi MySQL. Pentru fiecare cont se creează o nouă înregistrare în tabelui **mysql.user** şi nu se alocă acestui cont privilegii. Dacă contul există deja, va apărea o eroare.Fiecare nume de cont se va define astfel:

Exemplu:

```
CREATE USER 'utilizatorlab1'@'localhost';
```

Dacă se doreşte specificarea doar a numelui utilizatorului, fără host name, host name-ul va fi înlocuit cu '%'. Clauza IDENTIFIED BY se foloseşte doar în momentul în care se doreşte ca utilizatorul să se conecteze şi cu parole, dupa cum se poate observa în cele ce urmează:

Exemplu:

```
CREATE USER 'utilizatorlab'@'localhost' IDENTIFIED BY 'utcn';
```

MySQL consideră atât host name-ul cât şi user name-ul în momentul accesării serverului. Pentru a determina ce privilegii are un anumit cont, se va folosi clauza SHOW GRANTS.

Exemplu:

```
SHOW GRANTS FOR 'utilizatorlab'@'localhost';
```

- **Instrucţiunea ALTER USER**

```
ALTER USER user_specification
[, user_specification] ...
user_specification:
user PASSWORD EXPIRE
```

Clauza ALTER USER modifică contul MySQL şi are ca rezultat expirarea parolei utilizatorului.

Exemplu:

```
ALTER USER 'utilizatorlab'@'localhost' PASSWORD EXPIRE;
```

După ce parola unui utilizator a expirat, toate operaţiile cu conexiune la server ale acestui utilizator vor rezulta într-o eroare până când utilizatorul va folosi clauza SET PASSWORD pentru a stabili o nouă parola.

- **Instrucțiunea DROP USER**

```
DROP USER user [, user] ...
```

Clauza DROP USER şterge unul sau mai multe conturi MySQL.

Exemplu:

```
DROP USER 'utilizatorlab1'@'localhost';
```

DROP USER nu închide automat sesiunea deschisă a unui utilizator, ci are effect doar după închiderea sesiunii.

- **Instrucţiunea GRANT**

Sintaxa acestei instrucțiuni este prezentată în cele ce urmează:

```
GRANT privilege [ , privilege… ] ON bd TO user [ , user… ];
```

Se pot specifica următoarele privilegii pentru utilizatorii bazei de date:

- SELECT: citire coloane
- INSERT (col-name): inserare tuple cu valori non null în coloană.
- DELETE: ștergere de tuple.
- REFERENCES (col-name): definire de chei străine (în alte tabele) pentru a referi coloana specificată.

Clauza GRANT acordă privilegii conturilor de utilizator.De asemenea, această clauză specifica alte caracteristici ale conturilor ca de exemplu folosirea conexiunilor securizate şi limitează accesul la resursele serverului. Pentru a o folosi, utilizatorul trebuie să deţină privilegiile GRANT OPTION şi să aibă şi el privilegiile pe care le acordă.

În mod normal, administratorul bazei de date va crea înainte un utilizator folosind CREATE USER iar apoi îi va acorda privilegii cu GRANT.

Pentru a determina ce privilegii rezultă în urma operaţiei, se foloseşte SHOW GRANTS.

Tabel 1. Definirea privilegiilor

ALL [PRIVILEGES]	acordă toate privilegiile la un nivel de acces specificat cu excepţia GRANT OPTION
ALTER	permite folosirea clauzei ALTER TABLE
CREATE	permite crearea de baze de date şi tabele
CREATE VIEW	permite crearea şi modificarea vederilor
DELETE	permite folosirea clauzei DELETE
DROP	permite ştergerea bazelor de date, vederilor şi tabelelor
GRANT OPTION	permite ştergerea sau adăugarea de privilegii unui cont
INDEX	permite crearea şi modificarea indecşilor
LOCK TABLES	permite folosirea LOCK TABLES pentru tabele unde sunt privilegii SELECT
SELECT	permite folosirea clauzei SELECT
SHOW DATABASES	permite folosirea SHOW DATABASES
SHOW VIEW	permite folosirea SHOW CREATE VIEW
TRIGGER	permite operaţiile cu triggere
UPDATE	permite folosirea clauzei UPDATE
USAGE	este sinonim cu "no privileges"

- **Privilegii globale**

Privilegiile globale sunt aplicabile tuturor bazelor de date pe un server specificat. Pentru a acorda privilegiile globale, se foloseşte sintaxa ON *.*:

Exemplu:

```
GRANT ALL ON *.* TO 'utilizator_ie'@'localhost';
GRANT SELECT, INSERT ON *.* TO 'utilizator_ie1'@'localhost';
```

- **Privilegii pentru bazele de date**

Privilegiile pentru o bază de date se acordă tuturor obiectelor dintr-o anumită BD. Pentru a acorda privilegii la nivelul BD se foloseşte sintaxa ON *db_name*.*:

Exemplu:

```
GRANT ALL ON gestiune_laboratoare.* TO 'util'@'localhost';
GRANT SELECT, INSERT ON gestiune_laboratoare.* TO 'util'@'localhost';
```

Dacă se foloseşte sintaxa ON * şi a fost selectată în prealabil o bază de date, privilegiile vor fi acordate la nivelui de BD pentru acea bază de date. Dacă nu există o bază de date predefinită, va apărea o eroare.

- **Privilegii pentru tabele**

Priviegiile pentru tabele se aplică tuturor coloanelor dintr-o tabelă dată. Pentru a se acorda privilegii la nivel de tabel, se foloseşte sintaxa ON *db_name.tbl_name* . De exemplu daca am dori sa acordăm utilizatorului ie privilegii la tabelul laboratoare se va proceda astfel:

Exemplu:

```
GRANT ALL ON gestiune_laboratoare.laboratoare TO 'ie'@'localhost';
GRANT SELECT, INSERT ON gestiune_laboratoare.laboratoare TO 'ie'@'localhost';
```

Dacă se specific doar numele tabelului, privilegiile se vor acorda tabelului cu acel nume din baza de date selectată anterior. Dacă nu este specificată o bază de date, se va genera o eroare.

- **Privilegii pentru coloane**

Privilegiile cu privire la coloane se acordă unei singure coloane dintr-un tabel dat. Fiecare privilegiu la nivel de coloană trebuie să fie urmată de numele coloanei sau coloanelor între paranteze.

```
GRANT SELECT (col1), INSERT (col1,col2) ON mydb.mytbl TO 'someuser'@'somehost';
```

- **Instrucțiunea RENAME USER**

```
RENAME USER old_user TO new_user
[, old_user TO new_user] ...
```

Instrucțiunea RENAME USER redenumeşte un cont existent MySQL. Se va genera o eroare dacă contul vechi nu există sau contul nou creat există deja.

Exemplu:

```
RENAME USER 'utilizatorlab'@'localhost' TO 'utilizator'@'localhost';
```

Dacă se specifica doar numele utilizatorului, partea de host name este definită de '%'.

RENAME USER determină folosirea privilegiilor vechiului utilizator de către utilizatorul nou. Chiar dacă nu sunt invalidate, bazele de date şi obiectele create de vechiul utilizator vor fi introduse în programe stocate şi vor fi mai greu de recuperate. Acestea vor avea un anumit nivel de securitate.

- ## Instrucțiunea REVOKE

Instrucțiunea REVOKE permite administratorul de sistem să revoace privilegiile pentru diferite conturi MySQL.

Sintaxa acestei clauze se poate observa în cele ce urmează:

```
REVOKE
priv_type [(column_list)]
[, priv_type [(column_list)]] ...
ON [object_type] priv_level
FROM user [, user] ...
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM user [, user] ...
REVOKE PROXY ON user
FROM user [, user] ...
```

Exemplu:

De exemplu, pentru revocarea drepturilor de inserare intr-o baza de date pentru utilizatorul ie se va scrie urmatoarea linie de cod:

```
REVOKE INSERT ON *.* FROM 'utilizator_ie'@'localhost';
```

Pentru a revoca toate privilegiile se va folosi următoarea sintaxă, care şterge toate priviegiile globale, pentru baze de date, tabele, coloane pe care le au utilizatorii la care se face referire.[23]

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'utilizator_ie1'@'localhost';
```

Aplicații

Aplicația 1: Să se creeze o vedere denumită test care să conţină toate datele din tabelul funcţii din baza de date gestiune_laboratoare, apoi să se afişeze datele conţinute în această vedere.

Rezolvare

```
use gestiune_laboratoare;
create view test as select * from functii;
select * from test;
```

Aplicația 2: Să se creeze o vedere denumită labresp care să conţină denumirea laboratoarelor şi responsabilul pentru fiecare dintre acestea cu coloanele laborator, nume şi prenume şi să se afişeze apoi această vedere.

Rezolvare

```
create view labresp (laborator, nume, prenume)
as select lab.denumire, r.nume, r.prenume
from laboratoare lab, responsabili r
where lab.ID_responsabil=r.ID ;
select * from labresp;
```

Aplicația 3: Să se adauge la vederea labresp şi funcţia responsabilului şi să se afişeze vederea

Rezolvare

```
alter view labresp (laborator, nume, prenume, functie)
as select lab.denumire, r.nume, r.prenume, f.denumire
from laboratoare lab, responsabili r, functii f
where (lab.ID_responsabil=r.ID) and (r.ID_functie=f.ID) ;
select * from labresp;
```

Aplicația 4: Ştergeţi vederea test

Rezolvare

```
Drop view test;
```

Aplicația 5: Creaţi o vedere care să conţină laboratoarele şi aparatele care se găsesc în acesta denumită labap ordonate după denumirea laboratorului. Deoarece ambele coloane au numele denumire, se vor preciza alte denumiri pentru coloane

Rezolvare

```
create view labap(lab, aparat)
as select lab.denumire, ap.denumire
from laboratoare lab, aparatura ap, lab_ap a
where (a.ID_lab=lab.ID)and(a.ID_ap=ap.ID_aparatura)
order by lab.denumire;
select * from labap;
```

Aplicația 6: Să se creeze un index cu numele APx care să facă referire la coloanele denumire şi spec_tehn din tabela aparatura

Rezolvare

```
CREATE INDEX  APx ON aparatura(denumire, spec_tehn);
```

Aplicația 7: Să se afişeze toţi indecşii din tabela aparatura

Rezolvare

```
show index from aparatura;
```

Aplicația 8: Să se şteargă indexul APx din tabela aparatura

Rezolvare

```
drop index APx on aparatura;
```

Exerciții propuse

1. Creaţi o vedere denumită locatii care să conţină lucrările de laborator şi laboratoarele unde se efectuează aceste laboratoare
2. Să se creeze vederea aparate care să conţină denumirea aparatelor care au verificare=1
3. Modificaţi vederea locaţii astfel încât să conţină şi adresa laboratorului
4. Creaţi vederea ap care conţine numele aparatelor din lucrarea de laborator cu ID=2, care au data de expirare a garanţiei după '2011-11-05'
5. Să se şteargă vederea locaţii
6. Sa se creeze o vedere supraf care să conţină denumirea laboratoarelor şi suprafaţa acestora în dm^2 , ştiind că aceasta este în m^2
7. Să se creeze 2 indecşi unul pentru tabela funcţii care să facă referire la coloana denumire şi să fie denumit den şi al doilea pentru tabela laboratoare care să se refere la adrese cu denumire adr
8. Să se afişeze indecşii tabelei laboratoare
9. Să se şteargă indexul creat anterior pentru tabela funcţii