horizontal line

****

Baze de date - concepte fundamentale și exerciții practice

Suport de curs și îndrumar de laborator

| Alex Butean | Adrian Nistor | Alin Cruceat | Eduard Ghergu |
| --- | --- | --- | --- |

# Cuprins:

[**Cuprins:**](#_gps4kw371tha) **0**

[**1. Introducere în baze de date**](#_niig86j2hhro) **2**

[1.1 Curs - Aspecte teoretice](#_8xccb48nckg7) 2

[1.2 Laborator - Noțiuni practice](#_1m5ukf8d0rsp) 5

[**2. Structura bazei de date**](#_rlm1b1is14cm) **7**

[2.1 Curs - Aspecte teoretice](#_tij298va2r6) 8

[2.2 Laborator - Noțiuni practice](#_3smqnxlq1v27) 10

[**3. Operații principale și interogări simple**](#_1c8x7x6eo9hc) **14**

[3.1 Curs - Aspecte teoretice](#_mj7lqh8dytcz) 14

[3.2 Laborator - Noțiuni practice](#_obekxe7vlajv) 19

[**4. Funcții importante**](#_olgykmg6nuw2) **20**

[4.1 Curs - Aspecte teoretice](#_ryra2qwki5we) 20

[4.2 Laborator - Noțiuni practice](#_eid8klfpfqs) 22

[**5. Interogări avansate**](#_o01vf6rkwgj6) **23**

[5.1 Curs - Aspecte teoretice](#_6vckywel1wi3) 23

[5.2 Laborator - Noțiuni practice](#_92zwph2fe36s) 23

[**6. Preluare date din mai multe tabele**](#_d3cs3mivqhgj) **26**

[6.1 Curs - Aspecte teoretice](#_tgwvd8qmu3dx) 26

[6.2 Laborator - Noțiuni practice](#_3uw6dzmn4nxd) 26

[**7. Modalități de preluare a datelor**](#_z59evxc007bc) **32**

[7.1 Aspecte teoretice](#_vfs7kiwl8xrl) 32

[7.2 Laborator - Noțiuni practice](#_xnxsgdal8lbp) 33

[**8. Securitatea datelor**](#_il249t925e0q) **35**

[8.1 Curs - Aspecte teoretice](#_8wy185lfyihn) 35

[8.2 Laborator - Noțiuni practice](#_tg0i0hxw2b8b) 35

[**9. Mecanisme de automatizare**](#_dno0bgnfkay5) **36**

[**10. Big Data**](#_7idut541vlai) **42**

[**12. Integrare cu mediul de programare**](#_caduwgvs1bqa) **42**

[**12. Add topic**](#_cypy2c44dxez) **42**

[**13. Recapitulare**](#_p8f6eaw0yy7y) **43**

[**14. Examen / Colocviu**](#_c4xbqzidwwjx) **43**

# 

# 1. Introducere în baze de date

## 1.1 Curs - Aspecte teoretice

**Datele** sunt informații primare (numere, mărimi, relații) care pot fi stocate, procesate și supuse unor proceduri de prelucrare.

O **bază de date**, numită și bază de date electronică este orice colecție de date sau informații, care este special organizată pentru căutare rapidă și interconectare cu un software digital. Bazele de date sunt structurate pentru a facilita stocarea, recuperarea, modificarea și ștergerea datelor împreună cu diverse operațiuni de prelucrare a datelor.

Baze de date frecvent întâlnite:

* agendă telefonică
* registrul contabil
* tranzacții financiare
* lista de cumpărături
* rețele de socializare

**Nevoia de organizare** a condus la apariția bazelor de date și structurarea informației astfel încât să fie îndeplinite următoarele nevoi:

* încrederea în sistemele digitale se bazează pe capacitatea acestora de a stoca date persistente
* cantitățile mari de date au nevoie de indexare pentru acces rapid (data indexing) și de o modalitate facilă de vizualizare
* structurarea datelor este un proces de durată care necesită etape complexe (previzibile sau mai puțin previzibile) care trebuie executate standardizat și independent de mediul de stocare
* mecanismele de partajare (data sharing) trebuie să permită acces simultan pentru mai multe tipuri și clase de utilizatori
* încrederea că operațiile asupra datelor sunt înregistrate (logging), analizate din punct de vedere al securității (data security), salvate incremental (backup) și restaurante la nevoie (rollback and restore)

| **Evoluția bazelor de date** | |
| --- | --- |
| Sistemul de gestiune a fişierelor | 1950 - 1960 |
| Baze de date ierarhice | Baze de date de tip rețea | 1960 - 1970 |
| Baze de date relaționale | 1970 - prezent |
| Modele semantice și modele obiectuale a datelor | 1970 - 1980 |
| SQL devine limbaj standard de interogare | 1980 |
| Baze de date orientate obiect | 1990 - prezent |

**Exemple de sisteme de baze** de date folosite în sisteme moderne și platforme foarte cunoscute:

* Twitter - MySQL, Cassandra, and FlockDB
* LinkedIn - Oracle and MySQL
* StackOverflow - MSSQL
* Microsoft - MSSQL
* Reddit - PostgreSQL and Cassandra
* Youtube - Google Big Table
* Myspace - MSSQL
* Wikipedia - MySQL
* Siemens - IBM- DB2
* Amazon - Oracle (for web services)

**Bazele de date relaționale** au fost introduse de către Edgar Frank Codd. Conceptul relațional al datelor se bazează pe teoria matematică a relaţiilor.

**Modelul informational** este o reprezentare abstractă, formală a entităților. Acest model include proprietățile lor, relațiile și operațiunile care pot fi efectuate pe ele.

**Modelul de date** este definit la un nivel mai concret și include mai multe detalii. El este destinat dezvoltatorilor de software și include protocoale specifice.

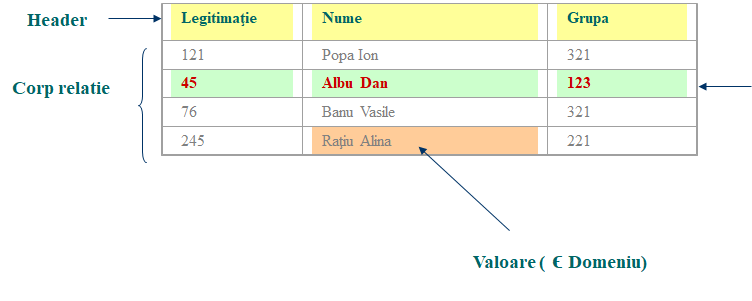
Structuri de date relaţionale:

* tuplu – set ordonat de valori care descriu anumite caracteristici ale datelor la un anumit moment în timp. Termenii uzuali: rând, articol, înregistrare
* atribut - evidenţiază o anumită caracteristică a datelor. Termeni uzuali: coloană, câmp
* domeniu - setul de valori pe care le poate lua un atribut. Toate valorile unui domeniu sunt de acelaşi tip
* entitate - un obiect real care există în realitate. Termeni uzuali: tabel, structura, obiect

În teoria bazelor de date, **o relație** este un set de tupluri (d1, d2, ..., dn), unde fiecare element din această mulțime este parte dintr-un domeniu de date.  
  
O relaţie R asupra domeniilor D1, D2, D3,…, Dn, nu neapărat distincte, este construită dintr-un header şi un corp al relaţiei.

**Header-ul** conţine un set fix de atribute A1, A2, A3, …, An, astfel încât fiecare atribut Ai este definit exact pe unul dintre domenii, şi anume Di, unde i=1,n

**Corpul** constă dintr-un set variabil în timp de tupluri, unde fiecare tuplu la rândul său constă dintr-un set de perechi atribut-valoare (Ai:vi), unde i=1,n. Există câte o astfel de pereche pentru fiecare atribut Ai din header. Pentru oricare pereche atribut-valoare (Ai:vi), vi este o valoare din domeniul unic Di asociat cu atributul Ai



**Gradul** unei relaţii este dat de numărul de atribute ale relaţiei. (Gradul relaţiei STUDENT este 3)

**Cardinalitatea** unei relaţii este dată de numărul de tupluri ale relaţiei. Cardinalitatea unei relaţii se modifică în timp. (Cardinalitatea actuală a relaţiei STUDENT este 4.)

**Universul relaţional** U(R) este alcătuit din toate combinaţiile posibile ale valorilor atributelor unei relaţii. Orice relaţie R este o submulţime a universului relaţional U(R).

**Proprietățile unei relații**:

* Tuplurile unei relaţii sunt neordonate
* Atributele unei relaţii sunt neordonate
* Toate valorile atributelor sunt atomice (indivizibile)
* Într-o relaţie nu există tupluri duplicat

**Tipuri de relații** între entități:

* unu la unu   
  (ex: relația între tabelele Persoană - Medic de familie o persoană are un singur medic de familie)
* unu la mai mulți | mai mulți la unu ( ex: relația între tabelele Client - Factură - Comenzile unui client generează mai multe facturi)
* mai mulți la mai mulți (ex: relația între tabelele Profesor și Student : mai mulți profesori predau pentru mai mulți studenți)
* referință proprie (ex: relația între tabelul Angajat și Angajat directorul unei companii este va fi propriul sau angajat și director)

Pentru un **studiu bibliografic** amănunțit și pentru extinderea cunoștințelor prezentate în acest material se recomandă următoarele materiale:

1. Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. 2008. Database Systems: The Complete Book (2nd. ed.). Prentice Hall Press, USA. [Download link](https://people.inf.elte.hu/miiqaai/elektroModulatorDva.pdf)
2. D. Lewis, 2016, Database systems, University of London. [Download link](https://london.ac.uk/sites/default/files/study-guides/database-systems.pdf)
3. C.J. Date, 2012, An Introduction to Database Systems 8th edition. [Download link](https://www.pdfdrive.com/an-introduction-to-database-systems-8e-by-c-j-datepdf-e31093920.html)
4. Neeraj Sharma, Liviu Perniu, Raul F. Chong, Abhishek Iyer, Adi-Cristina Mitea, Chaitali Nandan, Mallarswami Nonvinkere, Mirela Danubianu, Tradus de Liviu Perniu . 2010 Baze de date - Fundamente [Download link](ftp://public.dhe.ibm.com/software/dw/db2/express-c/wiki/database_fundamentals_ro_ro.pdf)
5. C.J. Date Database Design and Relational Theory: Normal Forms. Apress Publishing House. 2019. [Book link](https://www.oreilly.com/library/view/cj-dates-database/9781449336370/), [Download link](http://dl.booktolearn.com/ebooks2/computer/databases/9781449328016_database_design_and_relational_theory_890d.pdf)

## 1.2 Laborator - Noțiuni practice

MySQL Workbench este unul dintre cele mai folosite tool-uri la ora actuală pentru design, administrare creare și mentenanța bazelor de date.

Ceea ce ar trebui să înțelegeți inițial este că orice tool pentru programarea bazelor de date este format din 2 componente esențiale:

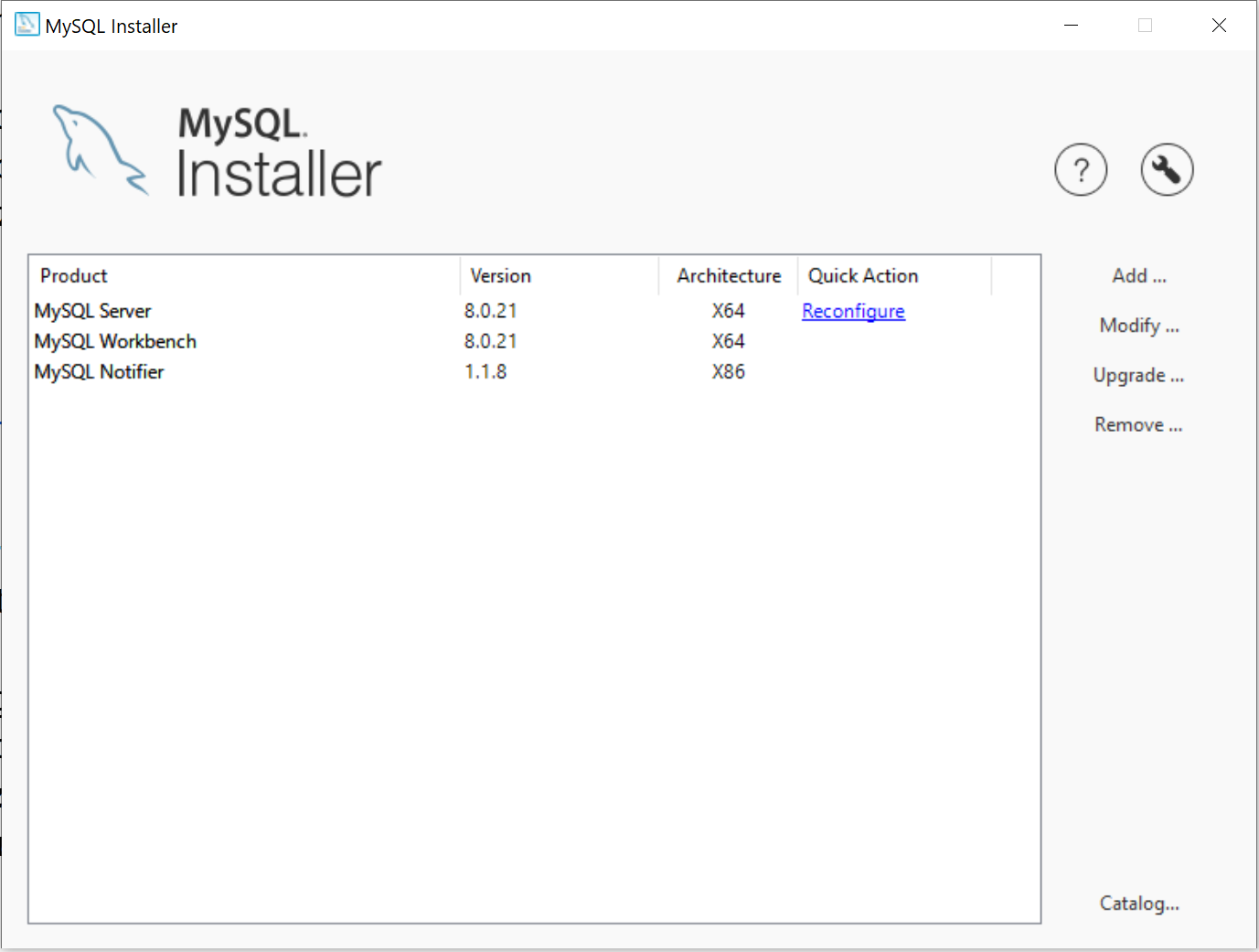
1. ***Server***: locul unde se stochează baza de date propriu-zisă. ,
2. ***Client:*** constă dintr-o aplicație locală (în cazul nostru MySQL Workbench) care comunică cu serverul printr-o conexiune de tip TCP/IP

Conexiunea dintre server și client de tip TCP/IP și are ca parametrii:

1. **Hostname:** adresa IP a serverului. Dacă serverul este local acest parametru va avea valoarea *“localhost”* sau *“127.0.0.1”*.
2. **Port:** - portul TCP/IP la care asculta serverul de MySQL. Prestabilit acesta are valoarea *3306,* dar în unele cazuri în care portul este ocupat de către altă aplicație este de recomandat să fie schimbat. Acest element este motivul pentru care dacă aveți mai multe servere de MySQL instalate, pornirea lor simultană este problematică, toate dorind portul ***3306.*** *se recomanda instalarea folosind alt port , ex:3307*

**Probleme frecvente:**

* există un server de baze de date deja instalat care rulează pe același port : recomandarea ar fi sa dezinstalati serverul existent sau să porniți serverele pe 2 porturi diferite
* aveți o versiune anterioara de MySQL Server instalat - se recomanda dezinstalarea si instalarea
* utilizatorul nu are drepturi de administrare pentru instalare
* componentele de client și server vin separat și trebuie selectate pentru instalare
* firewall-ul blochează accesul (pentru conexiuni la baze de date online)



Instalarea componentelor necesare pentru acest laborator se poate face direct din [MySQL Installer](https://dev.mysql.com/downloads/windows/installer/8.0.html), folosind opțiunea Add. La configurarea MySQL Server folosiți toate opțiunile prestabilite, exceptând portul dacă știți că mai aveți alte tool-uri pentru databases instalate (ex. Xampp, Wamp etc..)

Odată instalat tool-ul deschideți MySQL Workbench și creați-vă o conexiune nouă. Această conexiune vă va permite să vă conectați la server-ul pe care l-ați configurat anterior.

TIP: Dacă doriți ca altcineva să vă poată folosi serverul local, tot ceea ce are nevoie este de IP-ul vostru (Host name), de portul pe care ați configurat server-ul, hostname și password, precum le-ați configurat.

În laboratorul de azi veți folosi interfața MySQL Workbench. Încercați să o explorați și să vă familiarizați cu ea.



Unul din elementele pozitive legate de MySQL Workbench este că în momentul lucrului cu interfața grafică pentru creare editare sau ștergere de elemente, în momentul aplicării acestora va apărea un wizard ce va oferi codul MySQL ce va fi rulat pentru această procedură.

**Exerciții**

1. Folosiți [MySQL Installer](https://drive.google.com/file/d/1jrjrgTbS0RAjFDHazMw5WrSo3IV-d4Xj/) (varianta **completa** ~400 MB) pentru a instala componentele:
   * MySQL Workbench (Interfață - Client)
   * MySQL Server (Server)
   * MySQL Notifier (Utilitar)
2. Realizați conexiunea între client și server și creați o bază de date cu numele ***prima*** folosind interfața grafică.
3. Creați un tabel de test cu denumirea ***student*** cu structura: ***nr\_matricol***, ***nume***, ***prenume***, ***specializare .*** Alegeți tipurile de date corespunzătoare.
4. Populati tabelul ***student*** cu 10 înregistrări folosind interfața grafică.
5. Exersați funcționalitățile de ***import*** și ***export*** individual și apoi folosiți fișierele de la alți colegi.

# 2. Structura bazei de date

## 2.1 Curs - Aspecte teoretice

**Cheile (Keys)** sunt o parte importantă a unei baze de date relaționale și o parte esențială din structura tabelară. Acestea se asigură că fiecare înregistrare dintr-un tabel poate fi identificată în mod unic printr-unul sau o combinație de atribute. Acestea contribuie la consolidarea integrității și ajută la identificarea relației dintre tabele.

**Clasificarea cheilor** se realizează după următoarele caracteristici:

* rol în tabel:
  + Primary Key
  + Secondary Key (Alternate Key)
  + Foreign Key
  + Candidate Key
  + Super Key
* proveniența valorilor:
  + Natural Key
  + Surrogate Key
* componența atributelor
  + Simple Key ( un singur atribut)
  + Multi attribute Key ( mai multe atribute)
    - Compound Key
    - Composite Key

**Super Key** - o supercheie este orice combinație de câmpuri dintr-un tabel care identifică în mod unic fiecare înregistrare din acel tabel. Setul tuturor atributelor este o supercheie banală, deoarece în algebra relațională nu sunt permise rânduri duplicate.

**Candidate Key** - o cheie candidat este un subset minimal al unei superchei. O cheie candidată este un singur câmp sau cea mai mică combinație de câmpuri care identifică în mod unic fiecare înregistrare din tabel. Cea mai mică combinație de câmpuri distinge o cheie candidată de o supercheie. Fiecare tabel trebuie să aibă cel puțin o cheie candidată, dar în același timp poate avea mai multe.

**Primary Key** - o cheie primară se alege din mulțimea cheilor candidat și în funcție de context este cea mai potrivită pentru a fi cheia de referință pentru tabel. Cheile primare sunt obligatorii pentru fiecare tabel, fiecare înregistrare trebuie să aibă o valoare pentru cheia primară. Atunci când se alege o cheie primară din grupul de chei candidate, dacă este posibil, se alege întotdeauna o cheie simplă în defavoarea unei chei compuse din mai multe atribute.

**Foreign Key** - o cheie străină este (în general) o cheie primară dintr-un tabel care apare ca un atribute în alt tabel. Cu alte cuvinte, dacă am avea un tabel A cu o cheie primară X care ar fi legată de un tabel B unde X ar fi un câmp în B, atunci X ar fi o cheie străină în B. Cheile străine stabilesc relațiile dintre tabele.

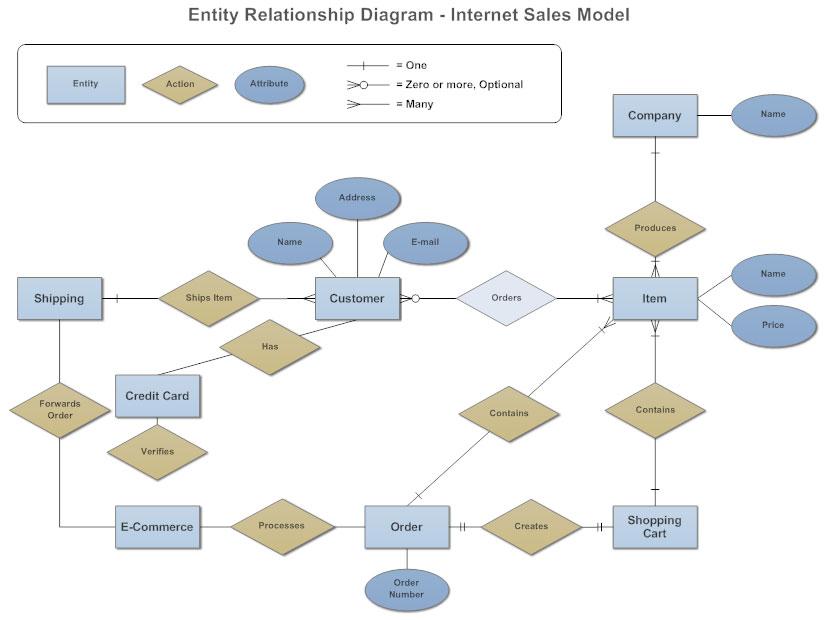
**Secondary Key** - cheile secundare sunt cheile candidat care datorită contextului nu au fost alese ca și cheie primară.

**Simple Key** - oricare dintre cheile descrise anterior pot fi formate dintr-unul sau mai multe atribute. O cheie simplă este cheia care formată dintr-un singur atribut.

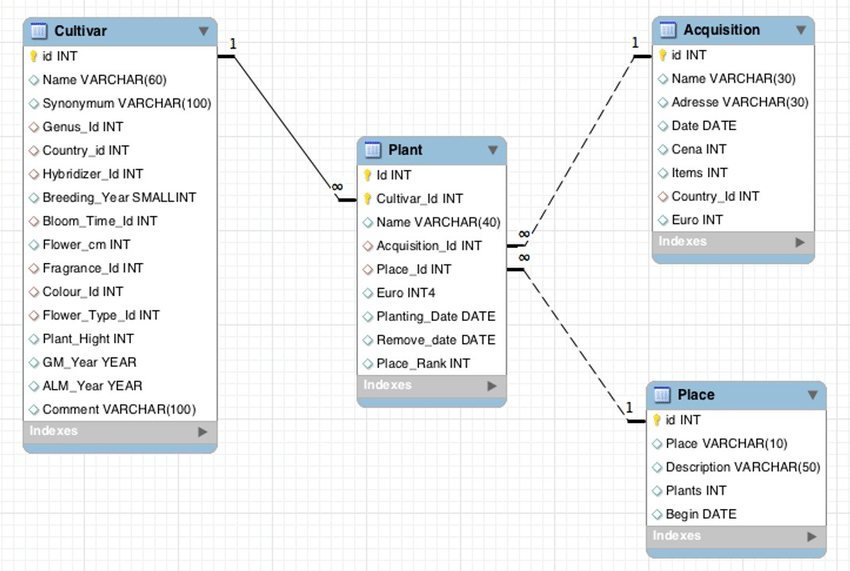
**Compound Key** - Cheia compound este formată din mai multe atribute simple și fiecare atribut simplu în parte este o supercheie.

**Composite Key** - Cheia composite este formată din mai multe atribute simple care, individual, nu sunt superkey și care doar luate împreună pot identifica unic o înregistrare.

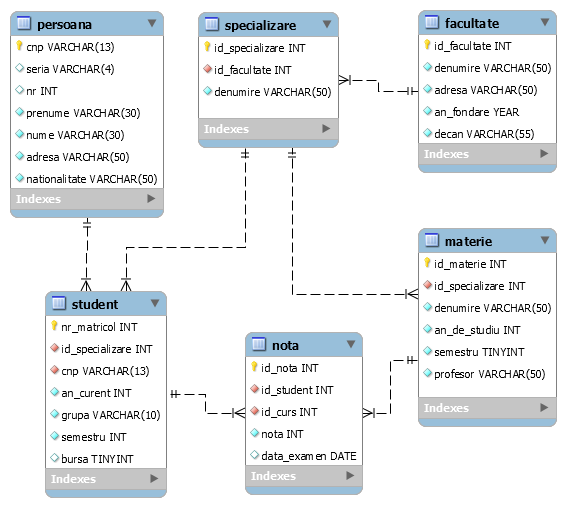
**Exemplu Diagrama - Model Entitate-relatie**



**Exemplu Diagrama - Model Relațional de date**

****

## 2.2 Laborator - Noțiuni practice

**Diagrama universitate (model relational)**  


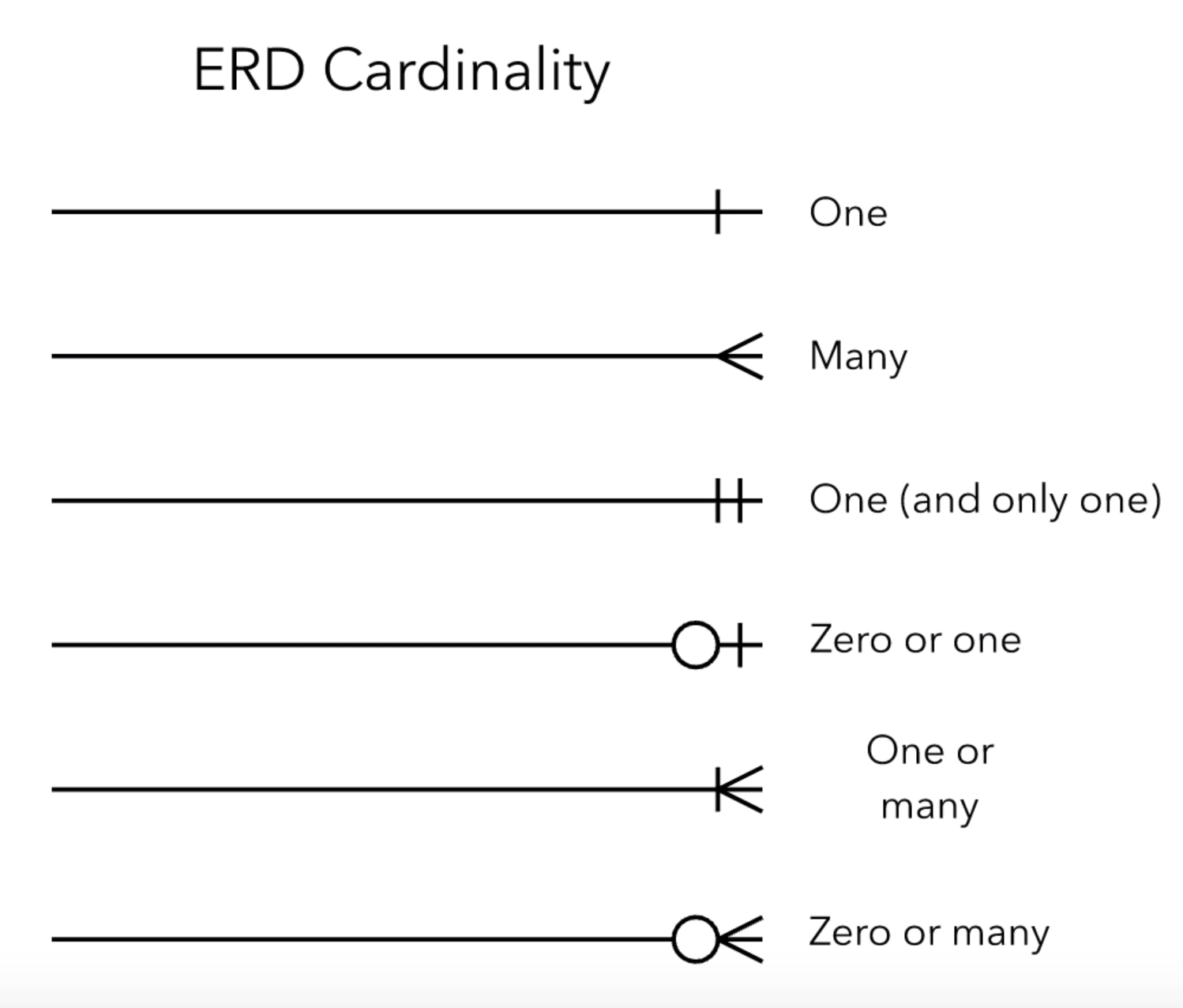
### 

**pentru studenți**

**Descarca fisierul SQL (pt import)- Click** [**AICI**](https://drive.google.com/file/d/1ahyKMP0ZHkne1wf0A4mA11-i7U531HCq/)

**doar pt profesori :)**

**Simbolurile ER (Entity-Relationship)**

****

**Simbolurile MySQL Workbench pentru coloane**

enter image description hereenter image description hereCheie: (Parte a) Primary Key

enter image description hereenter image description hereDiamant plin: NOT NULL

enter image description hereenter image description hereDiamant: NULL

enter image description hereenter image description hereenter image description hereColorat în roșu: (Parte a) Foreign key

enter image description hereenter image description hereDiamant albastru: Atribut simplu (fără cheie)

Pot exista combinații, cum ar fi:

enter image description here O cheie de culoare **roșie**, deci este Primary Key și este în același timp și Foreign Key

enter image description here O cheie de culoare galbenă, deci este numai Primary Key

enter image description here Un diamant plin, desenat cu albastru, deci este un atribut NOT NULL

enter image description here Un diamant plin, desenat cu roșu, deci este Foreign Key, NOT NULL

enter image description hereUn diamant albastru, deci este un atribut care poate fi NULL

enter image description here Un diamant roșu, deci este Foreign Key care poate fi NULL

**Convenții folosite pe parcurs**

Vom folosi câteva convenții pentru felul în care vom da numele tabelelor, coloanelor și a cheilor. Nu contează atât de mult convenția pe care o folosim, ci faptul că folosim cu toții aceeași convenție. Acest lucru ne va ușura alegerile pe care le facem pentru denumiri și mai important, ne va ușura codul sursa pe care îl vom scrie în eventualitatea unei aplicații care va interacționa cu baza noastră de date. (vor fi și exemple cand e nevoie).

* ***snake\_case*** pentru orice denumire : *aceasta\_este\_o\_denumire*
* denumirea entităților la singular: tabelul se numește *carte*, nu *cărți*
* ***id\_numele\_entitate*** oriunde avem o coloana care e cheie primară
* ***id\_numele\_entității\_referite*** pentru coloanele care reprezinta o cheie străină

**Crearea unei baze de date**

Comandă: CREATE DATABASE nume\_schema

CREATE DATABASE universitate;

**Crearea unui tabel**

Comandă: CREATE TABLE

CREATE TABLE universitate.facultate(  
 id INT,  
 denumire VARCHAR(50),  
 adresa VARCHAR(60),  
 an\_fondare YEAR  
)

**Modificarea unui tabel**

Putem folosi instrucțiunea de modificare a unei entități pentru a:

* adăuga, șterge sau modifica coloane de pe tabel
* adăuga, șterge sau modifica constrângeri de pe tabel

Comandă: ALTER TABLE

ALTER TABLE universitate.facultate  
ADD COLUMN decan VARCHAR(45) AFTER an\_fondare;

**Constrângeri (CONSTRAINTS)**

Putem folosi sistemul de management al unei baze de date (DBMS) pentru a ne asigura ca se satisfac anumite condiții pe datele pe care le avem.

Exemple de constrângeri:

* **NOT NULL** - Se asigură că valoarea pentru un atribut, nu poate avea valoarea NULL
* **UNIQUE** - Se asigură că valoarea pentru un atribut are o valoare unică între toate intrările din tabel.
* **PRIMARY KEY** - Combină NOT NULL și UNIQUE. Se asigura ca avem un identificator pentru fiecare intrare din tabel.
* **FOREIGN KEY** - Identifică intrări într-un alt tabel.
* **DEFAULT** - Se asigură că există o valoare default în cazul în care nu se trimite o valoare prin comandă.

**Adăugarea unei chei primare(Primary Key)**

Putem adauga o cheie primara în doua feluri:

1. Împreună cu instrucțiunea care crează tabelul:

CREATE TABLE universitate.specializare (  
 id\_specializare INT NOT NULL,  
 id\_facultate INT NULL,  
 denumire VARCHAR(45) NULL,  
 PRIMARY KEY (id\_specializare)  
);

1. Într-o instrucțiune care modifica tabelul:

ALTER TABLE universitate.specializare  
ADD PRIMARY KEY (id);

**Adăugarea unei chei străine(Foreign Key)**

Ca și pentru PRIMARY KEY, putem adăuga un FOREIGN KEY fie cand se creează tabelul, fie printr-o instrucțiune ALTER TABLE:

ALTER TABLE universitate.specializare  
ADD FOREIGN KEY (id\_facultate)  
REFERENCES universitate.facultate(id\_facultate);

**Exerciții**

1. Creați baza de date și toate tabelele conform schemei de mai sus.
2. Ce este AUTO\_INCREMENT și când îl folosim?
3. Explorați și aflați diferența dintre UNIQUE, PRIMARY și INDEX?
4. Explorați și aflați cum se folosește tipul de date ENUM.
5. Experimentați mesajele de eroare apărute în următoarele cazuri:
   1. inserare rând în tabel fără valoare pentru atributul cheie primară
   2. inserare rând în tabel fără valoare pentru un atribut cu constrângerea NOT NULL
   3. inserare rânduri în tabel cu valoare duplicată (non unică) pentru atributul cheie primară

# 3. Operații principale și interogări simple

## 3.1 Curs - Aspecte teoretice

Dintr-o perspectivă generală, comenzile de bază în procedurile de lucru cu procesarea datelor sunt : Create, Read, Update, Delete

Din perspectiva lucrului cu baze de date relaționale, comenzile de bază în procedurile de lucru cu procesarea datelor sunt : CREATE, ALTER, DROP, INSERT, UPDATE, DELETE, SELECT, WHERE, FROM, ORDER, LIMIT, BETWEEN, LIKE, IN

**Comanda CREATE**

Comanda Create este folosită pentru a crea un tabel în baza de date

CREATE DATABASE universitate;

CREATE TABLE universitate.specializare (  
 id\_specializare INT NOT NULL,  
 id\_facultate INT NULL,  
 denumire VARCHAR(45) NULL,  
 PRIMARY KEY (id\_specializare)  
);

**Comanda ALTER**

Comanda ALTER este folosită pentru a modifica (update) structura unui tabel care deja există în baza de date

ALTER TABLE universitate.specializare  
 ADD PRIMARY KEY (id);

**Comanda DROP**

Comanda DROP este folosită pentru a șterge complet un tabel.

DROP TABLE universitate.facultate

**Comanda INSERT**

Comanda INSERT este folosită pentru a adăuga(insera) un rând nou într-un tabel.

INSERT INTO universitate.specializare (id\_specializare, id\_facultate, denumire)   
VALUES ('100', '10', 'Literatura');

**Comanda UPDATE**

Comanda UPDATE este folosită pentru a actualiza (update) o înregistrare sau un set de înregistrări.

UPDATE universitate.specializare  
 SET denumire = "Literatura Romana"  
 WHERE id\_specializare=1;

**Comanda DELETE**

Comanda DELETE este folosită pentru a șterge o înregistrare sau un set de înregistrări.

DELETE FROM specializare  
 WHERE id\_specializare=1;

**Comanda USE**

Comanda USE setează baza de date default pe care se executa comenzi pentru a nu fi necesar să specificăm mereu denumirea tabelelor de tipul: baza\_date.nume\_tabel ci doar tabel.

USE university;

**Comanda SELECT**

Sunt rare aplicațiile ce folosesc baze de date în care nu predomină preluarea și afișarea de date. Odată introduse aceste date, vom dori să le analizăm în diferite moduri prin modificarea filtrelor, sortării și a calculelor pe care le aplicăm datelor brute. Vom folosi instrucțiunea SELECT pentru a selecta datele pe care le dorim returnate din baza de date în aplicația noastră. Deși poate fi utilizată pentru interogari simple, totuși, instrucțiunea SELECT are multe opțiuni și oferă o mare flexibilitate atunci când este utilizată în scenarii mai complexe. Scopul este să înțelegem structura și cele mai folosite opțiuni ale unei instrucțiuni SELECT, iar când situația o va impune, vor putea fi studiate și alte opțiuni ale acesteia.

O instrucțiune SELECT este compusă din mai multe elemente, fiecare începând cu un cuvânt cheie. Deși nu este necesar, pentru o perspectivă corectă asupra codului, convenția obișnuită este să scriem aceste cuvinte cheie cu majuscule și poziționate pe randuri diferite

* SELECT
* FROM
* WHERE
* ORDER BY

SELECT coloana1, coloana2

FROM tabel

WHERE <conditie>  
ORDER BY coloana3

Cea mai simplă formă a instrucțiunii SQL SELECT are doar 2 părți:

* coloanele pentru care dorim să obținem date (fără sa conteze ordinea lor
* tabela (ele) în care sunt definite acele coloane.

SELECT coloana1, coloana2

FROM tabel

Dacă dorim să preluăm toate informațiile disponibile în tabelă, putem folosi simbolul asterisc (\*) ca și modalitate de a specifica toate coloanele, iar interogarea noastră ar arăta de forma:

SELECT \*

FROM tabel

**Clauza WHERE**

Pe lângă selecție, putem și să limităm, sau să filtrăm, datele pe care dorim le preluăm din baza de date. Prin adăugarea unei clauze WHERE instrucțiunii SELECT, adăugăm una (sau mai multe) condiții pe care datele ar trebui să le îndeplinească pentru a fi selectate. În acest mod, numărul de înregistrări preluate din tabelă(e) va depinde de îndeplinirea condițiilor menționate. În multe cazuri, aici are loc cea mai mare parte a „acțiunii” unei interogări. În exemplul următor vom filtra datele, preluându-le numai pe cele care îndeplinesc condiția:

SELECT \*

FROM facultate

WHERE an\_fondare> 1950

**Clauza ORDER**

Până acum, am discutat despre filtrarea datelor, adică definirea condițiilor care determină ce înregistrări vor fi preluate din baza de date. Odată ce am stabilit ce coloane și rânduri (înregistrări) vor fi incluse în rezultatele interogării noastre SELECT, putem să controlăm și ordinea în sunt preluate înregistrările prin sortarea datelor.

Pentru a sorta înregistrările, vom include clauza ORDER BY. Aceasta va conține unul sau mai multe nume de coloane (separate prin virgulă), specificându-se astfel ordinea de sortare (după prima coloană, apoi după a doua ș.a.m.d.):

SELECT \*

FROM facultate

WHERE an\_fondare> 1950

ORDER BY adresa

**Este important de reținut că o coloană nu trebuie inclusă în lista coloanelor selectate (preluate) pentru a putea fi utilizată în clauza ORDER BY.**

ORDER BY sortează înregistrările în ordine crescătoare în mod implicit (ASC). Pentru a sorta înregistrările într-o ordine descrescătoare, putem utiliza cuvântul cheie DESC.

SELECT \*

FROM facultate

WHERE an\_fondare> 1950

ORDER BY adresa DESC

**Operatorul BETWEEN**

Operatorul BETWEEN selectează valori într-un interval. Valorile pot fi numere, text sau date.

SELECT denumire

FROM facultate

WHERE an\_fondare BETWEEN 2000 and 2010

**Operatorul LIKE**

Operatorul LIKE ne permite să utilizăm ‘wildcards’ pentru a specifica secvența de caractere pe baza căreia vor fi selectate înregistrările. Condiția LIKE este utilizată în clauza WHERE a unei instrucțiuni SELECT, INSERT, UPDATE sau DELETE.

Wildcards pentru LIKE :

* „%” este utilizat pentru a înlocui mai multe caractere
* „\_” este utilizat pentru a înlocui un singur caracter

Exemple:

* '%abc' - datele din coloana specificată trebuie să se termine cu secvența 'abc'
* 'abc%' - datele din coloana specificată trebuie să se înceapă cu secvența 'abc'
* '%abc%' - datele din coloana specificată trebuie să se conțină secvența 'abc'
* '\_bc%' - datele din coloana specificată trebuie să se conțină pe pozițiile 1 și 2 din string secvența 'bc' iar pe poziția 0 să fie orice alt caracter

SELECT decan

FROM facultate

WHERE decan LIKE '%escu'

**Operatorul IN**

Operatorul IN ne permite să verificăm apartenența la un set de valori într-o clauză WHERE.

SELECT denumire

FROM facultate

WHERE id\_facultate IN (1,10,9)

**Clauza LIMIT**

Clauza LIMIT este utilizată în instrucțiunea SELECT pentru a constrânge numărul de rânduri preluat. Clauza LIMIT acceptă unul sau două argumente:

* offset - specifică poziția primului rând care va fi preluat. offsetul primului rând este 0, nu 1.
* row\_count specifică numărul maxim de rânduri care vor fi preluate.Valorile ambelor argumente trebuie să fie întregi, pozitive, mai mari sau egal cu zero.

SELECT denumire

FROM facultate

LIMIT 1

sau

LIMIT 1, 2

SELECT

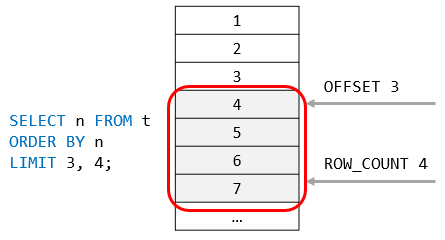
select\_list

FROM

table\_name

LIMIT [offset,] row\_count;

Următoarea imagine ilustrează clauza LIMIT:



Când utilizăm clauza LIMIT cu un singur argument, MySQL va utiliza acest argument pentru a determina numărul maxim de rânduri care trebuie returnate față de primul rând al setului de rezultate.

Prin urmare, următoarele sunt echivalente:

LIMIT row\_count;

și

LIMIT 0, row\_count;

## 3.2 Laborator - Noțiuni practice

1. Din punct de vedere al restricțiilor de foreign key Identificați ordinea corecta pentru a introduce date în tabelele din structura de tabele universitate.
2. Folosind atat interfața grafica cât și linia de comanda(query window), populează (cu ce informații vrei tu) structura de tabele universitate astfel:
   * 10 persoane
   * 10 studenți
   * 4 facultăți
   * 6 specializari
   * 20 note
   * 10 materii
3. Afișează denumirea și id\_facultate pentru facultățile care au id\_facultate impar.
4. Afișează naționalitatea și numele persoanelor care au in componenta numelui litera “a”.
5. Afișează grupa studenților care au bursa și sunt în semestrul 2.
6. Afișează lista tuturor numelor persoanelor, ordonate în ordine alfabetică.
7. Afișează lista cu numele primelor 10 persoane, ordonate în ordine alfabetică.
8. Afișează numele decanilor care conduc facultăți fondate între anii 1900,1995. Rezolvă exercițiul prin 3 metode.
9. Adaugă în tabela materie o înregistrare cu materia: Etica.
10. Șterge toate materiile care contin string-ul “tica”.
11. Afișează toate atributele pentru toate înregistrările din tabela specializare. Rezolvă exercițiul prin 2 metode.

# 4. Funcții importante

## 4.1 Curs - Aspecte teoretice

[**Funcțiile agregat**](https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html) returnează o singură valoare, calculată prin procesarea mai multor valori (a unui set / mulțime de valori). Funcțiile agregat ignoră valorile NULL dintr-un set.

AVG - Returnează media aritmetica a valorilor

COUNT - Returnează numărul de itemi dintr-un rezultat

MAX - Returnează valoarea maximă dintr-un set de valori

MIN - Returnează valoarea minima dintr-un set de valori

SUM - Returnează suma unui set de valori

Un lucru important de observat este că **funcțiile agregat** limitează rezultatul querry-ului la **un singur rând** sau un număr mai restrâns de rânduri în cazul grupării (vom vedea într-un laborator viitor).

[**Funcțiile de timp**](https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html) executa operații pe tipuri de date temporale

NOW - Returnează data și ora curentă a sistemului

CURDATE - Returnează data curentă a sistemului

CURTIME - Returnează ora curentă a sistemului

DATE - Extrage partea de dată calendaristică dintr-o expresie

DATE\_FORMAT - Afișează informațiile temporale în diferite formate

DATEDIFF - Returnează diferenta intre doua valorile temporale

[**Funcțiile matematice**](https://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html) permit execuția de operații matematice clasice. Toate funcțiile matematice returnează NULL în cazul unei erori.

ABS - Returnează valoarea absolută a unui număr.

SIN, COS, TAN, COT - Returnează valorile sinus, cosinus, tangentă, cotangentă

ACOS, ASIN, ATAN - Returnează valorile de tip arc dintr-un număr ( cuprins între -1 și 1)

PI - Returnează valoarea lui π(pi)

ROUND - Returnează un număr rotunjit, opțional se poate menționa numărul de zecimale

CEILING - Returnează un număr rotunjit în sus

FLOOR - Returnează un număr rotunjit în jos

RAND - Returnează un număr aleator între 0 (inclusiv) și 1 (exclusiv).

SIGN - Returnează semnul unui număr. Funcția returnează 1 atunci când valoarea argumentului este pozitivă, returnează -1 când valoarea argumentului este negativă și returnează 0 când valoarea argumentului este 0.

[**Funcțiile pe stringuri**](https://www.w3resource.com/slides/mysql-string-functions.php) permit execuția de operații specifice pentru tipuri de date text.

ASCII - Returnează valoarea numerică a primul caracter din string-ul primit ca parametru.

CHAR - Primește ca parametru o valoare întreagă și returnează caracterul aferent codului ASCII . Inversul funcției ASCII.

CONCAT - Returnează valoarea concatenată între mai multe string-uri primite ca și parametru

LENGTH - Returnează lungimea unui string

LEFT - Primește ca parametru o valoare întreagă și un un șir de caractere și returnează un subșir calculat până la offsetul indicat de valoarea întreagă (incepand din partea stângă a șirului)

RIGHT - Primește ca parametru o valoare întreagă și un un șir de caractere și returnează un subșir calculat până la offsetul indicat de valoarea întreagă (incepand din partea dreaptă a șirului)

SUBSTR sau SUBSTRING - Returnează un subșir de caractere aflat între doua valori întregi.

UPPER - Convertește toate caracterele dintr-un șir în caractere upper case.

## 4.2 Laborator - Noțiuni practice

**Operatorul AS**

Operatorul AS reprezintă un alias ce ne permite să denumim o anumită coloană în tabelul afișat de SELECT. Operatorul acesta își arată valoarea în cazurile în care dorim să ordonăm după o coloană creată cu ajutorul funcțiilor, permițându-le să evităm repetarea unei formule în query.

SELECT denumire,   
YEAR(CURDATE()) - an\_fondare) AS vechime

FROM universitate.facultate

ORDER BY vechime

**Exerciții**

**!Important!** Pentru exercițiile de la acest capitol, orice coloană ce reprezintă o formulă, funcție sau combinație de funcții va fi reprezentată **cu un alias** relevant.

1. Afișează media notelor tuturor studentilor.
2. Afișați pentru fiecare notă din tabelul **nota** timpul trecut de la acordarea notei până în prezent.
3. Afișați din tabelul **persoană** numele complet al studentului într-o singură coloană, și seria cu numărul de buletin în altă coloană. ( rezolvati prin doua moduri)
4. Afișați din tabelul **student** numărul de studenți înmatriculați și numărul total de persoane înmatriculate. Acest exercițiu reflectă posibilitatea unei persoane de a fi student la mai multe facultăți simultan.
5. Afișează numele decanilor de facultate cu caractere mari : Ex: POPESCU. (Fără să modifici în baza de date)
6. Afișează CNP-ul, numele persoanei, luna nașterii (numeric, ex: 08), luna nașterii( text, ex: August) pentru toate persoanele. (Extrage luna nasterii din CNP.
7. Afișează în format ASCII numele profesorilor care au in componenta numelui litera ‘a’. (opțional)
8. Adaugă litera ‘x’ în fața numelui decanilor care conduc facultăți ce au denumirea formată din mai puțin de 5 caractere. (opțional)

# 5. Interogări avansate

## 5.1 Curs - Aspecte teoretice

Vezi tabla interactiva - grupuri, condiții, logica de grupare.

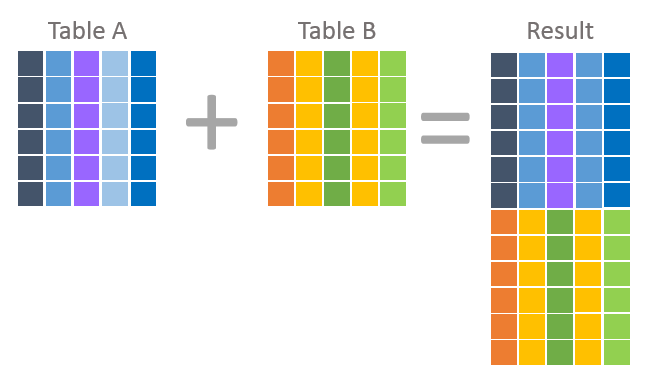
## 5.2 Laborator - Noțiuni practice

Noțiuni practice

**EXPLAIN** - oferă informații despre modul în care baza de date execută o interogare. În MySQL, EXPLAIN poate fi utilizat în fața unei interogări începând cu SELECT, INSERT, DELETE, REPLACE și UPDATE. Pentru o interogare simplă, ar arăta după cum urmează:

EXPLAIN   
SELECT \*   
FROM universitate.persoana

**UNION** este folosit pentru a combina setul de rezultate a două sau mai multe instrucțiuni SELECT.

Observați că fiecare instrucțiune SELECT din cadrul UNION trebuie să aibă același număr de coloane. Coloanele trebuie să aibă, de asemenea, tipuri de date similare. De asemenea, coloanele din fiecare instrucțiune SELECT trebuie să fie în aceeași ordine.

Operatorul UNION selectează în mod implicit numai valori distincte. Pentru a permite valori duplicate, utilizați cuvântul cheie ALL. ( UNION ALL)

În mod implicit, numele coloanelor din setul de rezultate ale UNION sunt de obicei egale cu numele coloanelor din prima instrucțiune SELECT din UNION.

O interogare UNION poate avea o singură clauză master ORDER BY.

SELECT profesor

FROM universitate.materie

UNION ALL

SELECT denumire

FROM universitate.specializare

**INTERSECT -** acest operator compară rezultatul a două interogări și returnează înregistrările (distincte)care sunt prezente atât de interogarea 1 cât și de interogarea 2. INTERSECT e un operator standard SQL, dar din păcate MySQL nu suportă acest operator.

Reguli de aplicare:

* Ordinea și numărul de coloane trebuie să fie aceleași
* Tipurile de date ale coloanelor corespunzătoare trebuie să fie compatibile

(SELECT nume

FROM universitate.persoana)

INTERSECT

(SELECT profesor

FROM universitate.materie);

**GROUP BY** - este folosit pentru a colecta date din mai multe înregistrări și a grupa rezultatul pe una sau mai multe coloane. Pe coloana/ coloanele targetate este foarte util a se aplica funcții agregat.

SELECT grupa, COUNT(nr\_matricol)

FROM universitate.student

GROUP BY grupa

**HAVING** - este folosit pentru filtrare (similar clauzei WHERE) și adaugare de condiții exclusiv pentru coloane pentru care se face gruparea.

SELECT grupa, COUNT(nr\_matricol)

FROM universitate.student

GROUP BY grupa

HAVING grupa=’230’

**Exerciții**

1. Experimentați folosirea operatorului EXPLAIN în câteva (min 4) din exercițiile rezolvate la temele anterioare.
2. Enumerați toate facultățile și specializările în același set de rezultate. Afișați denumirea lor și valoarea cheii principale. Ordonați rezultatul într-o manieră alfabetică inversată în funcție de denumirea specializarii.
3. Afișați toate notele distincte din baza de date și numărul de studenți care au luat notele respective. ( Ex: 5 (nota), 200 (studenți) ). Formatați rezultatul astfel:

| Nota | Nr studenți care au luat nota |
| --- | --- |
| 5 | 200 |

1. Identificați persoanele care fac parte din aceeași familie (au același nume și locuiesc la aceeași adresa. Grupați înregistrările din tabela persoane după nume și adresă. Afișați doar persoanele care locuiesc în București. Inserati în tabela **persoana** înregistrări relevante pentru exemplificarea corectă a scenariului. Formatați rezultatul astfel:

| Nume familie | Adresa familie | Număr membri familie |
| --- | --- | --- |
| Popescu | Strada Roșie, Nr 3, București | 3 |

1. (Opțional) Găsiți variante de a simula operatorul INTERSECT folosind alți operatori care se regasesc in MySQL.

# 6. Preluare date din mai multe tabele

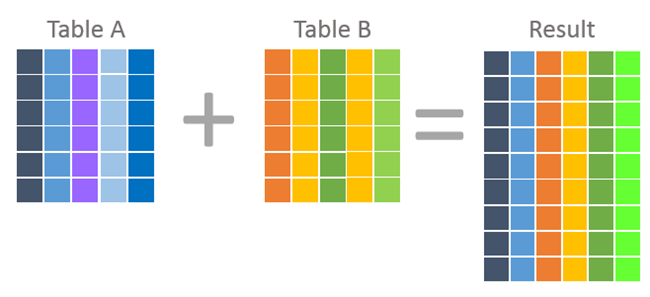
## 6.1 Curs - Aspecte teoretice

conspect individual

## 6.2 Laborator - Noțiuni practice

Clauza Join ne permite să combinăm înregistrările a 2 tabele aflate în relație. Scopul acestei clauze este aducerea în același tabel (rezultat) a tuturor intrărilor din 2 tabele având ceva în comun.

Utilizarea JOIN-ului este relativ simplă, după FROM specificând tabelul ce vrem să îl combinăm, adăugând apoi un ON unde specificăm criteriul de combinare al tabelelor.



SELECT \*

FROM table\_A

JOIN table\_B

ON table\_A.thing = table\_B.thing

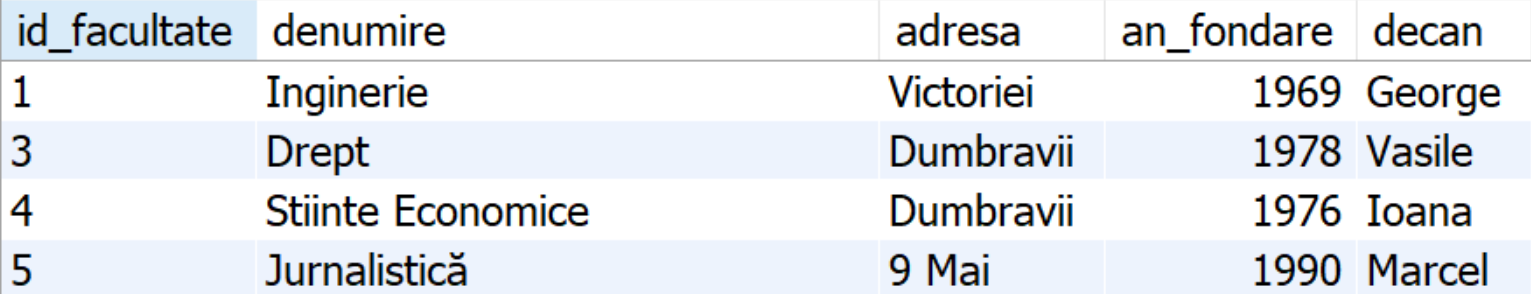
JOIN-ul are și o formă simplificată, în care doar înșirăm tabelele în FROM, adăugând criteriile de combinare într-o clauză WHERE ce arată astfel:

SELECT \*

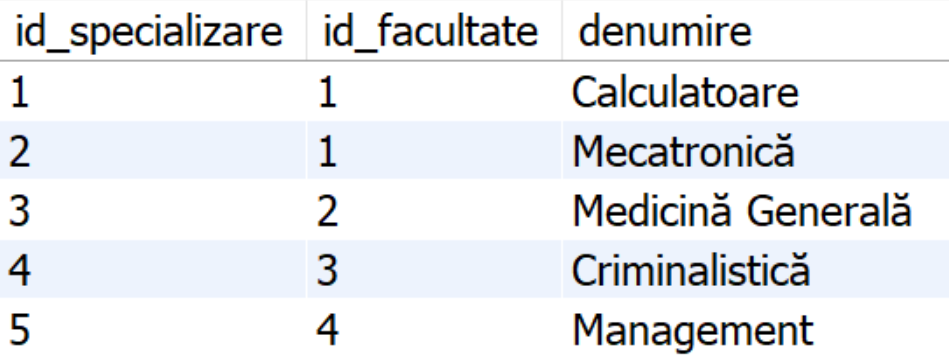
FROM table\_A, table\_B

WHERE table\_A.thing = table\_B.thing

Considerând că tabela noastră facultate este populată în felul următor:

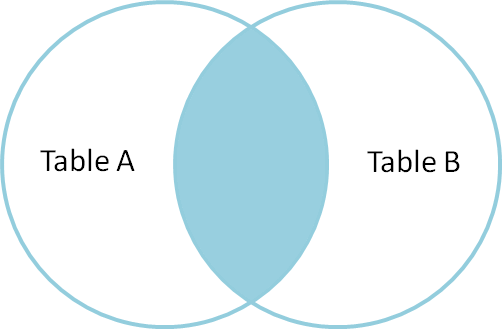


iar tabela noastră specializare astfel:



rulând un query cu o clauză Join vom căuta să afișăm denumirile facultăților și specializărilor în același tabel.

**INNER JOIN** sau pur și simplu **JOIN** va combina toate intrările care au un corespondent în celălalt tabel.



SELECT facultate.denumire AS nume\_facultate,

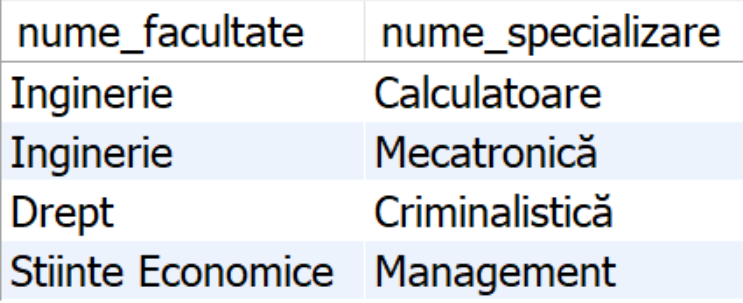
specializare.denumire AS nume\_specializare

FROM facultate

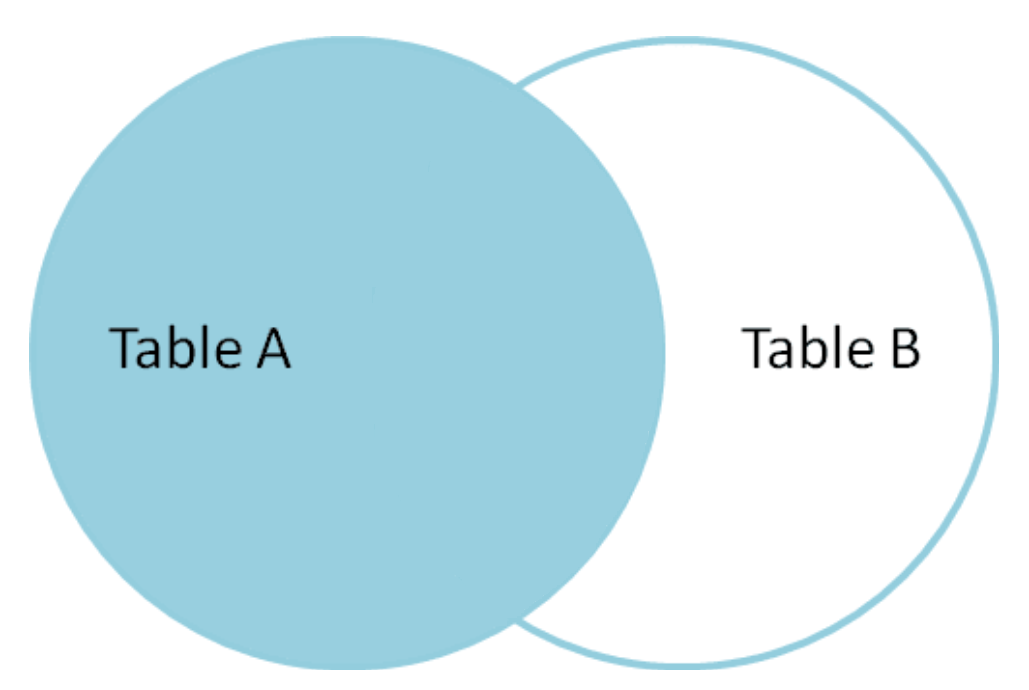
JOIN specializare

ON facultate.id\_facultate = specializare.id\_facultate

Rezultatul obținut va fi:



**LEFT JOIN** ne va afișa ***TOATE*** datele din prima tabelă cu corespondentul lor din cea de-a 2-a tabelă, dacă există, iar dacă nu cu corespondent NULL.



SELECT facultate.denumire AS nume\_facultate,

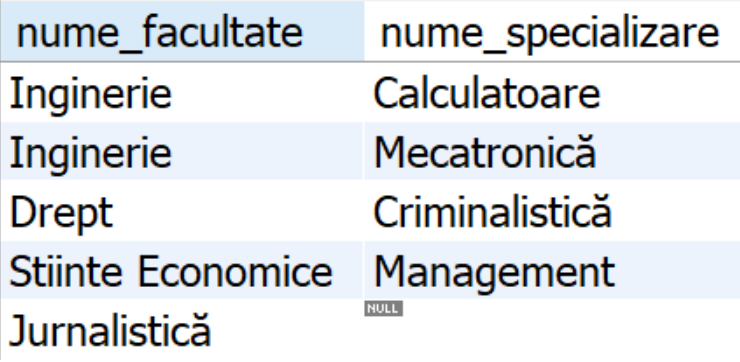
specializare.denumire AS nume\_specializare

FROM facultate

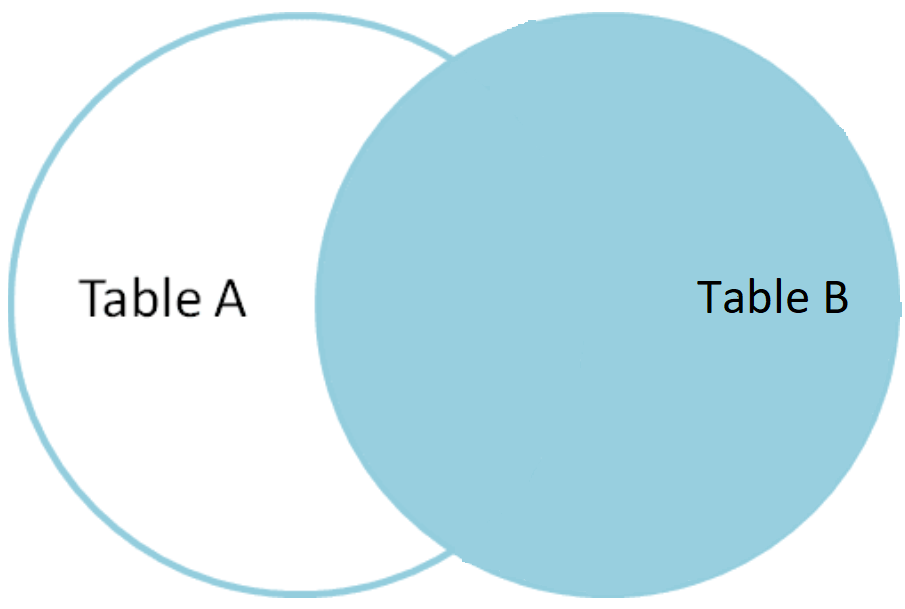
LEFT JOIN specializare

ON facultate.id\_facultate = specializare.id\_facultate

Rezultatul obținut va fi:

****

**RIGHT JOIN** ne va afișa ***TOATE*** datele din cea de-a 2-a tabelă cu corespondentul lor din prima tabelă, dacă există, iar dacă nu cu corespondent NULL



SELECT facultate.denumire AS nume\_facultate,

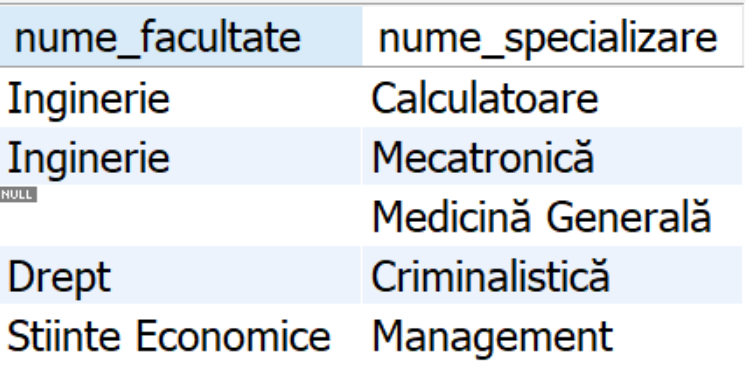
specializare.denumire AS nume\_specializare

FROM facultate

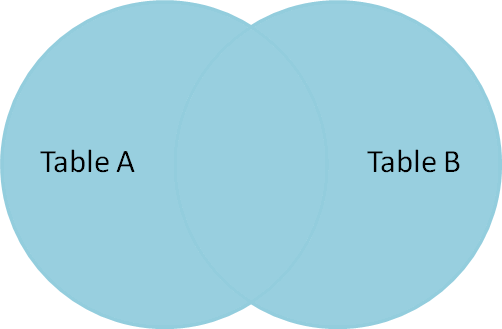
RIGHT JOIN specializare

ON facultate.id\_facultate = specializare.id\_facultate

Rezultatul obținut va fi:

****

**FULL (OUTER) JOIN** - ne va afișa ***TOATE*** datele din ambele tabele cu sau fără corespondenți



**Această clauză nu este disponibilă în platforma Workbench**

SELECT facultate.denumire AS nume\_facultate,

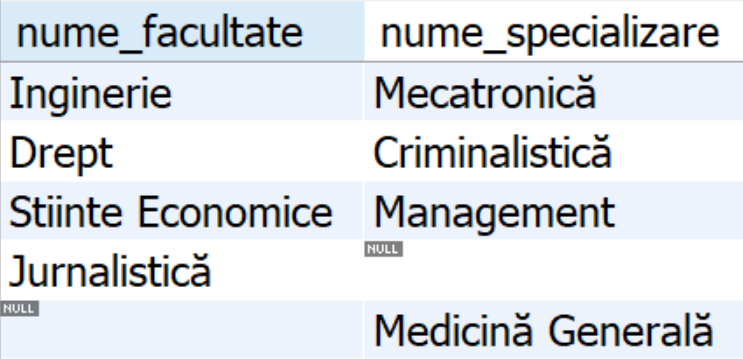
specializare.denumire AS nume\_specializare

FROM facultate

FULL JOIN specializare

ON facultate.id\_facultate = specializare.id\_facultate

Rezultatul obținut va fi:

****

**CROSS JOIN** - este modul în care putem face un ***cross product*** în MySQL

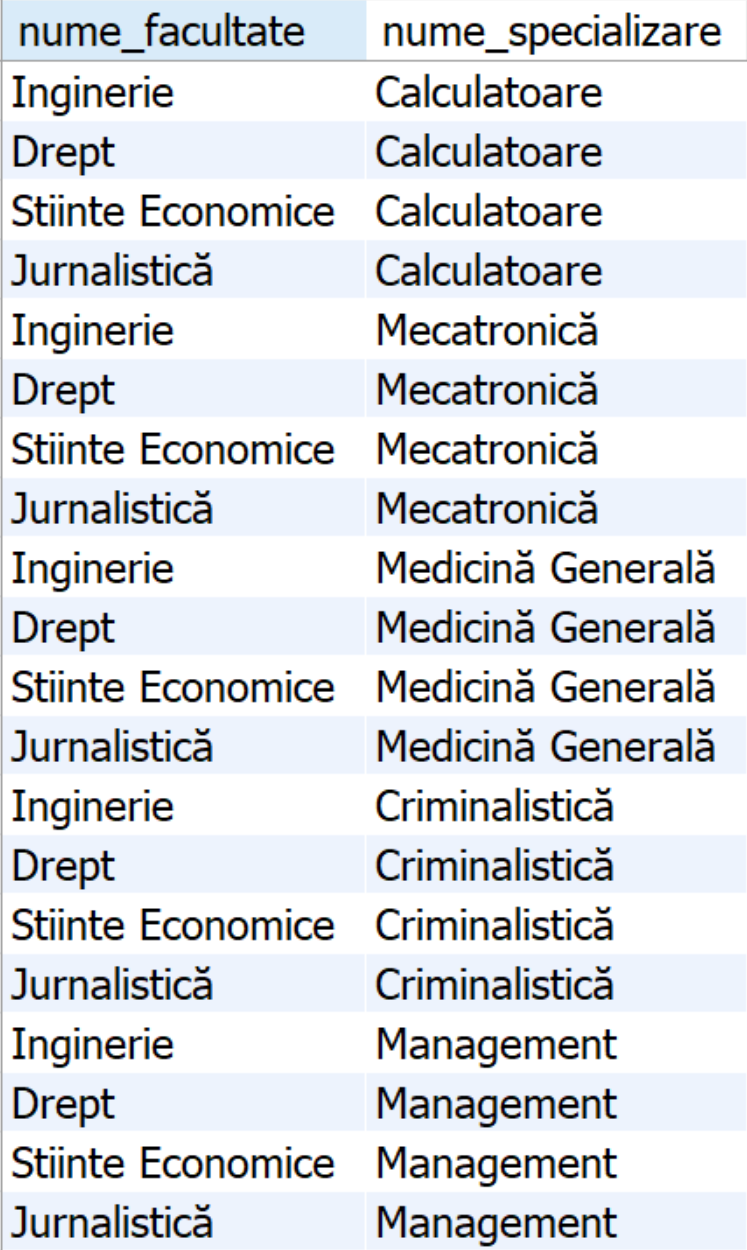
SELECT facultate.denumire AS nume\_facultate,

specializare.denumire AS nume\_specializare

FROM facultate

CROSS JOIN specializare

Rezultatul obținut va fi:



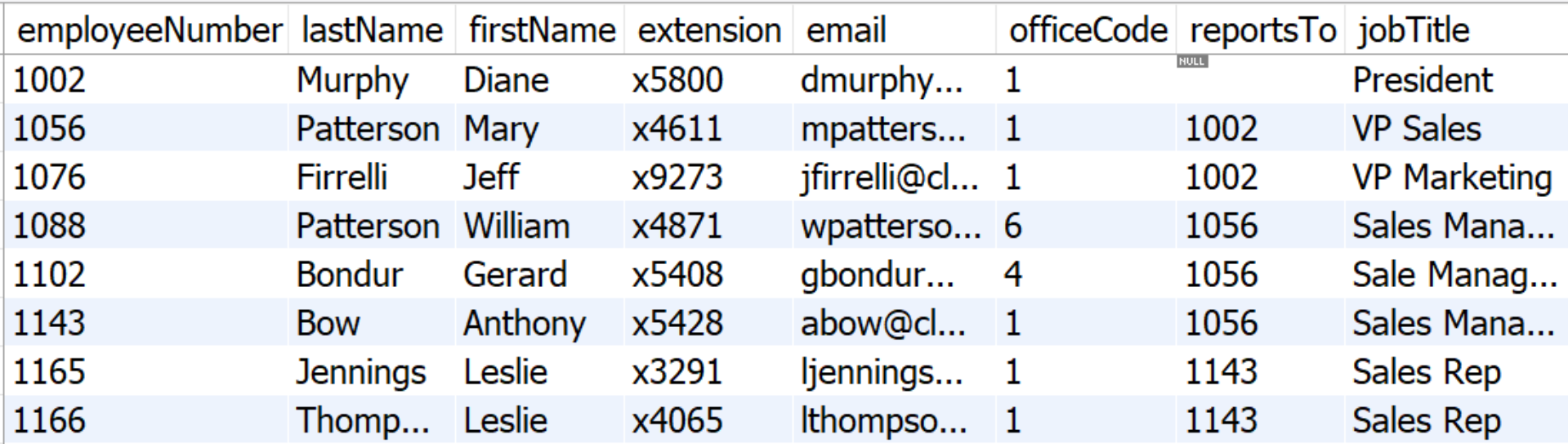
Pentru ca toată noțiunea de JOIN să fie mai ușor de înțeles hai să luăm și un exemplu puțin mai funny :))



**Alias-ul de tabel**

Acum că vom începe să lucrăm cu multiple tabele simultan vom întâlni cazuri în care va trebui să dăm JOIN pe același tabel. Luați următorul caz:

Avem o tabelă *employees* unde *employeeNumber* este cheia primară, iar *reportsTo* este self referencing foreign key. Putem întâlni cazuri în care ni se cere să afișăm numele angajatului și numele superiorului în același tabel.



Rezolvarea acestui caz se face astfel:

SELECT CONCAT(A.firstName, " ", A.lastName) AS angajat,

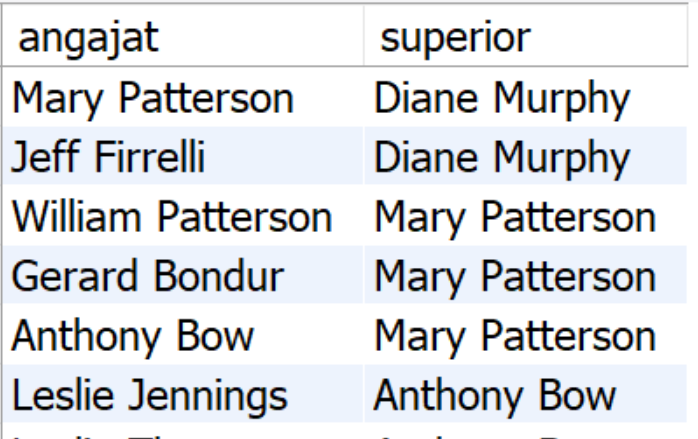
CONCAT(B.firstName, " ", B.lastName) AS superior

FROM employees A

JOIN employees B

ON A.reportsTo = B.employeeNumber;

Rezultatul obținut va fi:



Ce am făcut aici, din cauză că trebuie să importăm același tabel employees de 2 ori, ar fi apărut confuzie legat de unde luăm reportsTo și employeeNumber și la cine ne referim (angajat sau superior) când ne legăm de firstName și lastName.

Cu această metodă ne setăm ca A să fie instanța de care reprezintă angajat, iar B instanța care reprezintă superior.

Acest principiu se folosește de obicei în clauza JOIN deoarece ne permite să reducem numele tabelei la ceva mai scurt astfel încât nu vom repeta un nume lung de tabel de fiecare dată când trebuie să specificăm de cine aparține un atribut.

**Exerciții**

1. Listează ID-ul și numele specializărilor de la facultatea X (alege tu string-ul X).
2. Afișează numele specializării care are cele mai multe cursuri. (opțional, rezolvati prin doua metode)
3. Afișează un set de informații care să conțină următoarele:

* numărul matricol al studentului
* numele studentului
* prenumele studentului
* numele specializării

1. Afișează un set de informații care să conțină următoarele:

* nota
* numele și prenumele studentului (într-o singură coloană)
* numele materiei pentru care a fost primită nota

1. Afișează numele și codul matricol al studentilor din anul 2 care au cel puțin o notă mai mică decât 5.
2. (Opțional) Având noțiunile din acest laborator precum și din laboratoarele trecute, creați un query care să vă afișeze rezultatul unui FULL OUTER JOIN (în ciuda faptului că această clauză nu este suportată).

# 7. Modalități de preluare a datelor

## 7.1 Aspecte teoretice

partitionare verticala, orizontala, replicare, clusterizare

<https://www.digitalocean.com/community/tutorials/understanding-database-sharding>

<https://engineering.fb.com/2020/08/24/production-engineering/scaling-services-with-shard-manager/>

<https://stackoverflow.com/questions/4594738/clustering-sharding-or-simple-partition-replication>

<https://dzone.com/articles/challenges-of-sharding-mysql>

<https://medium.com/pinterest-engineering/sharding-pinterest-how-we-scaled-our-mysql-fleet-3f341e96ca6f>

<https://blog.yugabyte.com/how-data-sharding-works-in-a-distributed-sql-database/>

## 7.2 Laborator - Noțiuni practice

O subinterogare este de fapt o interogare de sine stătătoare care activează în interiorul unei interogări principale.

O subinterogare poate fi integrată în oricare din zonele: SELECT, FROM, INSERT, UPDATE, DELETE, WHERE, SET, etc.

O subinterogare se mai numește și o interogare interioară ( inner query) iar interogarea principală se mai numește și interogare master / exterioară ( outer query).   
  
Într-o logică firească (matematică) a execuției operațiilor , mai întâi se execute interogările interioare și apoi cele exterioare ( similar execuției bazate pe paranteze din calculul matematic).

De foarte multe ori folosirea subinterogărilor este mai puțin optimă decât folosirea unor JOIN-uri dar subinterogările sunt mai ușor (mai uman) de citit și de înțeles.

În contextul subinterogărilor este foarte importantă folosirea următoarelor seturi de parametrii / comenzi:

* operatori matematici de comparație (=, !=, >, <, <=, >=)
* condiții de comparație (IN, ANY, ALL)

Toți parametrii de mai sus sunt aplicabili atât la nivelul mulțimilor de valori (elemente redate cu virgulă) cât și la nivelul rezultatelor unor subinterogări.

Operatorul IN permite verificarea apartenenței unei valori la o mulțime.

SELECT prenume

FROM persoana

WHERE nume IN ("Popescu", "Ionescu", "Popa");

Afișează prenumele persoanelor al căror nume se regaseste in lista respectivă.

SELECT prenume

FROM persoana

WHERE CNP IN (SELECT CNP FROM STUDENT)

Afișează prenumele persoanelor care sunt studenti inmatriculati.

SELECT prenume

FROM persoana

WHERE CNP NOT IN (SELECT CNP FROM STUDENT)

Afișează prenumele persoanelor care sunt în baza de date dar NU sunt studenti inmatriculati.

SELECT id\_specializare, denumire

FROM specializare

WHERE id\_facultate IN (SELECT id\_facultate FROM facultate WHERE denumire="x")

Listează ID-ul și numele specializărilor de la facultatea X (alege tu string-ul X). Ex 1 de la capitolul anterior.

Operatorul ANY returnează adevărat dacă oricare dintre valorile unei mulțimi îndeplinesc condiția.

SELECT prenume

FROM persoana

WHERE CNP = ANY (SELECT CNP FROM STUDENT)

Afișează prenumele persoanelor care sunt studenti inmatriculati.

Operatorul ALL returnează adevărat dacă toate valorile subinterogării îndeplinesc condiția.

SELECT nota

FROM nota

WHERE nota > ALL (SELECT nota FROM nota WHERE data\_examen=curdate())

Pentru baza noastră de date nu se poate evidentia un exemplu logic cu operatorul ALL, astfel încât cerința ne-logică, dar acceptata este : Afișează notele care sunt mai mari decat toate notele primite azi.

Mai multe exemple cu operatorii menționați mai sus se găsesc :[AICI](https://dev.mysql.com/doc/refman/8.0/en/comparison-operators.html), [AICI](https://dev.mysql.com/doc/refman/8.0/en/any-in-some-subqueries.html), [AICI](https://www.tutorialspoint.com/What-is-the-use-of-ALL-ANY-SOME-IN-operators-with-MySQL-subquery),

Exerciții

Rezolvați exercițiile de mai jos exclusiv folosind subinterogari:

1. Listează ID-ul și denumirea specializărilor care aparțin facultăților fondate înainte de anul 1920.
2. Setează data examenului anul viitor, de ziua Europei, pentru toate notele corespunzătoare materiilor predate în anul 2 de studiu.
3. Șterge toate persoanele care nu au adresa completată corect (NULL sau mai mică de 10 caractere) și nu sunt studenți înmatriculați.
4. Afișează un set de informații care să conțină următoarele:

* numărul matricol al studentului
* numele studentului
* prenumele studentului
* numele specializării

1. Afișează numele și codul matricol al studentilor din anul 2 care au cel puțin o notă mai mică decât 5.
2. (Opțional) Anul acesta, de Ziua Aviaţiei Române, Profesorul Hermann Oberth vrea să le dea tuturor studenților săi câte o nota aleatoare (8, 9 sau 10) la toate materiile sale. Ca să-l ajuți ai putea sa faci un insert magic :) care să-i îndeplinească dorința.

# 8. Securitatea datelor

## 8.1 Curs - Aspecte teoretice

Securitatea bazei de date cuprinde o serie de măsuri de securitate concepute pentru a proteja sistemul de gestionare a bazelor de date (SGBD) de atacuri externe, acces nepermis la date și reducerea timpului de disponibilitate(uptime).

Tipurile de măsuri de securitate a bazei de date includ:

1. protejarea infrastructurii adiacente care găzduiește baza de date (rețeaua și serverele) - detalii [AICI](https://phoenixnap.com/kb/server-security-tips), [AICI](https://www.unixmen.com/security-measures-protect-servers/), [AICI](https://www.lightedge.com/blog/data-center-security-guide/)
2. configurarea sigură a SGBD - detalii [AICI](https://www.dbta.com/Editorial/News-Flashes/IOUG-Insight-5-Best-Practices-for-Securing-Databases-101930.aspx), [AICI](https://docs.oracle.com/cd/B19306_01/network.102/b14266/checklis.htm#i1010098), [AICI](https://security.berkeley.edu/education-awareness/best-practices-how-tos/system-application-security/database-hardening-best),
3. mecanisme sigure de backup - detalii [AICI](https://sqlbak.com/blog/backup-and-recovery-best-practices), [AICI](https://docs.microsoft.com/en-us/biztalk/core/best-practices-for-backing-up-and-restoring-databases), [AICI](https://www.tutorialride.com/dbms/database-backup-and-recovery.htm), [AICI](https://www.veritas.com/information-center/data-backup-and-recovery)
4. proceduri de audit - detalii [AICI](https://www.sqlshack.com/sql-server-auditing-best-practices/), [AICI](https://www.dnsstuff.com/server-auditing-windows-sql-file-servers-auditing), [AICI](https://www.computerweekly.com/tip/Best-practices-for-audit-log-review-for-IT-security-investigations)
5. proceduri de migrare - detalii [AICI](https://dzone.com/articles/10-best-practices-for-data-migration) , [AICI](https://www.etlsolutions.com/new/data-migration-best-practice-a-practical-guide/), [AICI](https://www.dataversity.net/five-data-migration-best-practices-ensure-successful-move/)
6. nivele de acces - detalii [AICI](https://www.hostinger.com/tutorials/mysql/how-create-mysql-user-and-grant-permissions-command-line), [AICI](https://www.dummies.com/programming/sql/sql-user-access-levels/), [AICI](http://appcentral.centraldecompras.pb.gov.br/help/help85_client.nsf/f4b82fbb75e942a6852566ac0037f284/94a7a71fefb689778525760900121741!OpenDocument),
7. funcții de criptare / decriptare detalii [AICI](https://digitalguardian.com/blog/what-data-encryption), [AICI](https://medium.com/searchencrypt/what-is-encryption-how-does-it-work-e8f20e340537), [AICI](https://www.precisely.com/blog/data-security/data-encryption-101-guide-best-practices), [AICI](https://docs.microsoft.com/en-us/azure/security/fundamentals/data-encryption-best-practices)
8. procese compatibile cu normele legislative în vigoare (GDPR) - [AICI](https://gdpr.eu/what-is-gdpr/), [AICI](https://dzone.com/articles/5-ways-to-make-your-database-gdpr-compliant), [AICI](https://www.itproportal.com/features/questions-database-admins-should-ask-about-gdpr/), [AICI](https://www.365villas.com/getting-your-database-gdpr-compliant/)

## 8.2 Laborator - Noțiuni practice

Recapitulare

Exerciții

Rezolvă exercițiile de mai jos prin ce metodă dorești (join-uri, subinterogari, variante mixte) cu condiția să ai **cel puțin un exercițiu** în care să nu folosești join-uri deloc ( nici explicite, nici implicite).

1. Afișați numele și prenumele studentilor care au numele similar cu numele unui profesor. În baza de date, numele profesorului este un singur câmp, dar conține 2 câmpuri logice (prenume și prenume) iar ordinea lor nu este definită. Găsiți soluții inteligente pentru a rezolva această problemă.
2. E sfarsit de an, secretara vrea să șteargă din baza de date toți studenții care nu au fost sau nu au luat nici un examen. Afișează codul matricol și id-ul facultății al acelor studenți. Rezultatul trebuie sortat după denumirea facultăților.
3. Hai sa vedem care sunt cei mai buni studenți. Afișează numele, prenumele, facultatea și media notelor primilor 10 studenți din fiecare facultate. Afișează rezultatul sortat după criterii:
   1. denumire facultății
   2. denumire specializarii
4. Hai să vedem dacă profesorii respecta regulile. Afișează numele profesorilor care în ultimii 5 ani au stabilit examene în afara perioadelor de sesiune (1 Feb - 10 Martie | 10 Iunie - 8 Iulie).
5. (opțional) Extrage din baza de date toate cuvintele (nume, denumiri, adrese, etc) care conțin litera “a”.
6. (opțional) Universitatea vrea sa trimită mesaje de 8 Martie pentru toate doamnele și domnisoarele din mediul academic. Din păcate, nu avem nici o informație despre identitatea de gen a studenților. Trebuie să afișăm prenumele și numele tuturor persoanelor de sex feminin.  
   Hint-uri :
   1. Dacă prenumele se termină cu „a” este o șansă mare ca acea persoană să fie femeie
   2. prima cifra din CNP variază mereu de la femei la bărbați, reușești să găsești o regulă?
   3. corelați cele doua hint-uri de mai sus 2 și vedeți dacă aveți același rezultat
   4. încercați să vă ajustați interogarea cu condiții suplimentare
   5. dacă e prea simplu, sigur ți-a scăpat ceva :)

# 9. Mecanisme de automatizare

Aspecte teoretice

1. Tranzacții (Transactions) - prezentări studenți : [LINK1](https://docs.google.com/presentation/d/1d66c_YzxzS8o7JoPGvovn33fwHaxotobiWoIGD8qqiQ/) , [LINK2](https://docs.google.com/presentation/d/1RLkLehU-wMx05MGDf365ZE7A6L4H-teHB9CzLJP9_5U/)
2. Proceduri stocate (Stored procedures) - prezentări studenți : [LINK1](https://docs.google.com/presentation/d/1uNGhgTiuETR4UFRyJ7a1_pxZCADhDPelPZshUS_X3FA/) , [LINK2](https://docs.google.com/presentation/d/1utbmyr16mEhHwktp6-Gb4XeSfD1NZ6scGBG6JAjXjXs/)
3. Declanșatoare (Triggers) - prezentări studenți : [LINK1](https://docs.google.com/presentation/d/1p6mn5q1gyccqjK8HxJPz9kSkWXSjWGMOuXwd5jNZ-VE/) , [LINK2](https://docs.google.com/presentation/d/1_jyxLR_T9XZUq1r2gg7vrSGHWUzYoE641bbJtGKlAQ4/)

Noțiuni practice

1. **Tranzacții**

În MySQL sunt implementate mecanismele necesare operațiilor tranzacționale. Mai multe informații [aici](https://www.w3resource.com/mysql/mysql-transaction.php).

Pentru a iniția o tranzacție, vom folosi instrucțiunea START TRANSACTION. BEGIN sau BEGIN WORK sunt aliasuri pentru START TRANSACTION.

Pentru a finaliza (commit) o tranzactie aflată în derulare și a face persistente modificările efectuate în cadrul său, vom folosi instrucțiunea COMMIT.

Pentru a anula (rollback) o tranzactie aflată în derulare și a renunța la modificările efectuate în cadrul său, vom folosi instrucțiunea ROLLBACK.

În scopul activării sau dezactivării opțiunii de a face persistente modificările efectuate în cadrul unei tranzacții în mod automat (auto-commit), vom utiliza instrucțiunea SET AUTOCOMMIT. Implicit, MySQL face persistente modificările efectuate în cadrul unei tranzacții în mod automat.

Pentru dezactivarea persistenței automate:

SET autocommit = 0;

sau

SET autocommit = OFF;

Pentru activarea persistenței automate:

SET autocommit = 1;

sau

SET autocommit = ON;

**Exemplu COMMIT:**

USE universitate;

SET AUTOCOMMIT = OFF;

START TRANSACTION;

INSERT INTO persoana VALUES (‘1901010323823’, 'SB', 444567, 'Vasile', 'Popescu', 'Str. Strada, Nr. 34', 'Romana');

INSERT INTO student VALUES (456, 12, ‘1901010323823’, 2020, 312, 1, 0);

COMMIT;

**Exemplu ROLLBACK:**

USE universitate;

SET AUTOCOMMIT = OFF;

START TRANSACTION;

DELETE FROM persoana;

ROLLBACK;

1. **Proceduri Stocate**

O procedură stocată este o secvență de cod SQL care este salvată în SGBD, și care poate fi apelată de câte ori este necesar. Spre deosebire de interogările (queries) ad-hoc (cele pe care le edităm și rulăm din consolă sau MySQL Workbench), procedurile stocate vor fi analizate de către SGBD după ce au fost create, fiind creat un plan de rulare a acestora, lucru ce permite o performanță ridicată și previne apariția unor erori accidentale.

Procedurile stocate pot avea sau nu parametri și pot întoarce zero sau mai multe rezultate în urma rulării lor.

**Exemplul 1:**

DELIMITER //

CREATE PROCEDURE `TotiStudentii` ()

BEGIN

SELECT \* FROM universitate.student;

END

// DELIMITER ;

Pentru apelarea unei proceduri stocate fără parametri:

CALL universitate.TotiStudentii();

Rezultatele rulării procedurilor stocate pot fi modificate prin furnizarea unor argumente la momentul apelării acestora. Însă, pentru ca acest lucru să fie posibil, este necesar ca o procedură stocată să aibă definiți parametrii corespunzători. Aceștia pot fi parametri de intrare (IN), ieșire (OUT), sau ambele (INOUT).

**Exemplul 2:**

DELIMITER //

CREATE PROCEDURE `StudentDupaCNP` (

IN cnpStudent VARCHAR(13)

)

BEGIN

SELECT \* FROM universitate.student

WHERE cnp = cnpStudent;

END

// DELIMITER ;

Pentru apelarea unei proceduri stocate cu parametri:

CALL universitate.StudentDupaCNP(‘1901010323823’);

1. **Funcții**

Asemenea ca în orice limbaj de programare avem posibilitatea de a crea funcții care să ne ajute să evităm repetarea anumitor formule sau anumitor structuri de cod care ne returnează un produs bazat pe alți parametri. Funcțiile diferă în schimb față de proceduri în ideea că funcțiile nu lucrează cu tabele și nu au cunoștință de absolut nimic ceea ce un tabel conține ci lucrează strict cu date.

În MySQL avem deja predefinit un set de funcții precum LENGTH, MAX, MIN, AVG, CONCAT, DATEDIFF care primind ca parametrii strict date returnează un element bazat pe datele de intrare introduse.

**Exemplu:**

CREATE FUNCTION full\_name ( f\_nm VARCHAR(20),

l\_nm VARCHAR(20))

RETURNS VARCHAR(40) DETERMINISTIC

RETURN CONCAT(f\_nm, " ", l\_nm);

O declarație de funcție conține:

* unul, mai mulți sau nici un parametru
* RETURNS în care specificăm tipul de date returnat urmat de DETERMINISTIC (dacă funcția returnează același rezultat la mai multe apeluri ale funcției cu aceiași parametri) sau NOT DETERMINISTIC (în cazul funcțiilor de tip random)
* RETURN care specifică concret ce returnează funcția

Odată declarată funcția se apelează ca orice altă funcție predefinită de MySQL:

SELECT full\_name(nume, prenume)

FROM persoana

**4. Declanșatoare**

În MySQL, un declanșator (trigger) este un set de instrucțiuni, stocat la nivelul SGBD, apelat automat ca răspuns la un eveniment cum ar fi INSERT, UPDATE, sau DELETE care apare la nivelul tabelului specificat.

Standardul ANSI SQL definește două tipuri de declanșatoare: la nivel de înregistrare (row-level) și la nivel de instrucțiune (statement-level).

Un declanșator definit la nivel de înregistrare este rulat de fiecare dată când o înregistrare este adăugată (inserted), modificată (updated), sau ștearsă (deleted). De exemplu, dacă o tabelă are 100 de înregistrări adăugate, modificate sau șterse, declanșatorul este rulat automat de 100 de ori.

Un declanșator la nivel de instrucțiune este rulat o singură dată pentru fiecare tranzacție, indiferent de câte înregistrări sunt adăugate (inserted), modificate (updated), sau șterse (deleted)

MySQL suportă numai declanșatoarele la nivel de înregistrare.

CREATE TRIGGER trigger\_name

{BEFORE | AFTER} {INSERT | UPDATE| DELETE }

ON table\_name FOR EACH ROW

trigger\_body;

Pentru a face diferența între valorile coloanelor Anterior (BEFORE), respectiv Ulterior efectuării modificărilor, vom utiliza modificatorii NEW, respectiv OLD.

Spre exemplu, dacă actualizăm valoarea coloanei persoana.nume a unei înregistrări, în trigger\_body, putem accesa valoarea existentă a numelui folosind OLD.nume anterior actualizării, iar noua valoare folosind NEW.nume.

DELIMITER $$

CREATE TRIGGER nota\_mare

BEFORE INSERT

ON nota FOR EACH ROW

IF NEW.nota<10

THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Domnule profesor va rugam sa ne dati doar note de 10';

END IF $$

DELIMITER ;

Exerciții

1. Unul dintre studenți a fost nevoit să-și schimbe actul de identitate, ocazie cu care s-a observat și faptul că CNP-ul a fost introdus greșit la înmatriculare. Actualizați datele despre studentul respectiv asigurându-vă că toate modificările sunt efectuate cu succes.

**Indicație**: Pentru a dezactiva verificările pe care SGBD le face apropo de integritatea referențială (sunt verificate relațiile între tabele), se poate folosi variabila FOREIGN\_KEY\_CHECKS după cum urmează:

dezactivare: SET FOREIGN\_KEY\_CHECKS = 0;

activare: SET FOREIGN\_KEY\_CHECKS=1;

1. La secretariat este nevoie de o varianta simpla de afla media generala a studentilor pentru fiecare an universitar (finalizat). Folosind proceduri stocate, faceți acest lucru posibil.
2. Secretara de departament este responsabilă de raportarea real-time și trebuie sa faca rapoarte zilnice cu top 10 cei mai buni studenți din anul universitar curent. Munca ei ar fi foarte mare dacă ar trebui sa se uite peste toate notele mereu. Ca sa o ajutam, am putea sa folosim un trigger, astfel de fiecare data cand un student primește o nota, ea trebuie sa primească o situația recalculată cu media tuturor notelor obținute pe anul curent pt top 10 cei mai buni studenți.
3. (opțional) Compuneți câte 1 exercițiu (cerință) care sa necesite rezolvarea folosind funcții, tranzacții, triggers și proceduri stocate.

# 10. Big Data

Aspecte teoretice

Prezentare ISM + TI - Document de lucru - [LINK](https://docs.google.com/presentation/d/1JAdBcwyS2DNumUcDaSZcKcny1Obsl-AFBlo5PIkHBak/)

Prezentare Calculatoare - Document de lucru - [LINK](https://docs.google.com/presentation/d/16gT6Jf7OudlvdnhwdiME9YqL6EYxzPtrdqFlwlLNjPo/)

Prezentere profesor - [LINK](https://docs.google.com/presentation/d/1SbJFz_0jjCocA3RfTYG5nFuFIU7Q5Lm7YNdVXouvVNs/)

Noțiuni practice

adăuga text aici

Exerciții

adăuga text aici

# 12. Integrare cu mediul de programare

Aspecte teoretice

adăuga text aici

Noțiuni practice

adăuga text aici

Exerciții

adăuga text aici

# 12. Add topic

Aspecte teoretice

adăuga text aici

Noțiuni practice

adăuga text aici

Exerciții

adăuga text aici

# 13. Recapitulare

Aspecte teoretice

adăuga text aici

Noțiuni practice

adăuga text aici

Exerciții

adăuga text aici

# 14. Examen / Colocviu

Aspecte teoretice

adăuga text aici

Noțiuni practice

adăuga text aici

Exerciții

adăuga text aici