

Curs 9

2022/2023

# Programarea aplicațiilor web

- Programarea aplicațiilor web
  - An V RC
    - 1.5C/1L/1P

# Program

- An V
  - Saptamana 1
    - Luni 17-20 Curs (Intro/HTML/CSS)
  - Saptamanile 2-8
    - Luni 16(17)-18 Curs
    - Luni 18-20 Laborator
  - Saptamanile 9-14
    - Luni 16(17)-18 Curs - **online**
    - Luni 18-20 Proiect - **online**

# Orar

- <https://orar.etti.tuiasi.ro/> : C->16(17)-18, L/P -> 18



FACULTATEA DE ELECTRONICA, TELECOMUNICATII SI TEHNOLOGIA INFORMATIEI

55RC

ETTI\_

	1 8:00 - 8:50	2 9:00 - 9:50	3 10:00 - 10:50	4 11:00 - 11:50	5 12:00 - 12:50	6 13:00 - 13:50	7 14:00 - 14:50	8 15:00 - 15:50	9 16:00 - 16:50	10 17:00 - 17:50	11 18:00 - 18:50	12 19:00 - 19:50
L										Damian R. PAW (C) 2.13 TC (R)	Damian R. PAW (L) 2.13 TC (R)	
Ma								C1	Scripcariu L. RCALSC (C)		Scripcariu L. RCALSC (L) 2.13 TC (R)	
Mi								Casian-Botez I. Etic (C) P6 (Amf.)	Casian-Botez I. Etic (S) P6 (Amf.)		Trifina L. TEFO (L) 3.25 TTI (L) Alecsandrescu I. POO (L) CI5(Corp C)	
J									Sirbu A. POO (C) P2 (Amf.)		Trifina L. TEFO (L) 3.25 TTI (L)	
V											Trifina L. TEFO (C) 3.25 TTI (L)	

# Nota

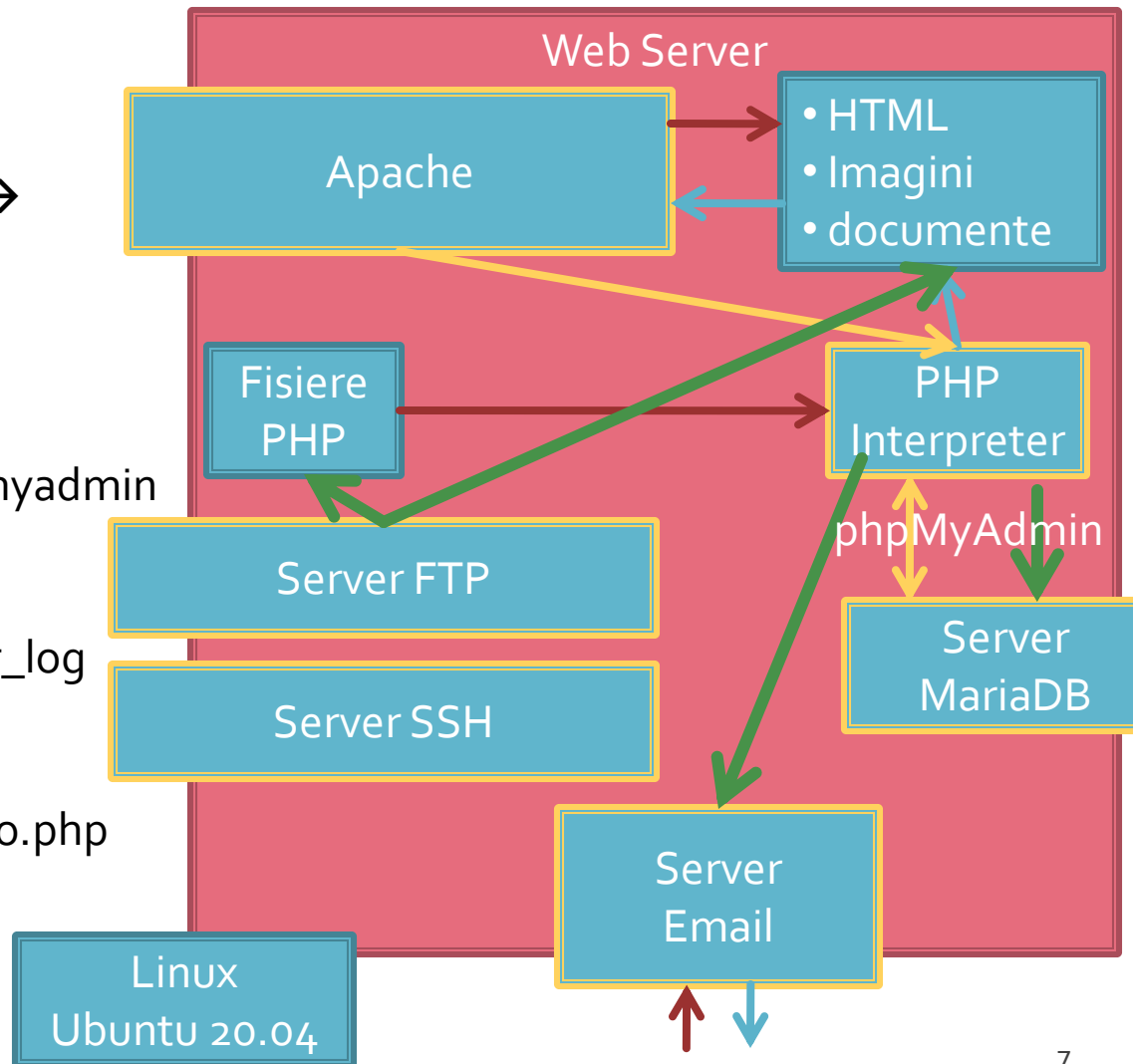
- An V
  - 33% E
  - 66% Aplicatii
    - 33% L
    - 33% P

# Nota

- An V
  - 33% E
  - 33% L
  - 33% P
- Laborator - Prezenta
  - 1pz = 1p ( $p > 5 \leftrightarrow$  Curs)
- Examen
  - Prezenta la curs: 3pz = 0.5p
  - Asemanator cu materia de proiect
- Activitate suplimentara
  - Dupa terminarea activitatii la laborator
  - +2 **1**p la E/L

# Utilizzare LAMP

1. login → **paw**:masteretti
2. ifconfig → 192.168.30.5
3. putty.exe → 192.168.30.5 → SSH → **paw**:masteretti (remote login)
4. [alte comenzi linux dorite]
5. FTP → Winscp → SFTP → student:masterrc@192.168.30.5
6. MySql → http://192.168.30.5/phpmyadmin → **root**:masteretti
7. Apache Error Log →
  - 7a. putty → nano /var/log/httpd/error\_log
  - 7b. http://192.168.30.5/logfile.php (nonstandard)
8. PHP info → http://192.168.30.5/info.php



# Tema bonus

- logfile.php
  - Afiseaza log Apache (erori php majore)
- ~~1p 2p~~ **suplimentar** la laborator/examen
- Modificare logfile.php pentru a afisa **toate** erorile PHP
  - php.ini – activare erori ✓
  - php.ini – locatie erori ✓
  - logfile.php – afisare log PHP ✓
  - drepturi de acces la fisiere (Linux)



# Server referinta LAMP

- Linux, doua variante
  - Centos 7.1
    - PHP 5.4.16
    - MariaDB 5.5.44 / root:masterrc
    - Apache 2.4.6
    - **root**/student:masterrc
  - **Ubuntu** 20.04
    - PHP 7.4.3
    - MariaDB 10.3.31 / root:masteretti
    - Apache 2.4.41
    - **paw**/student:masteretti
    - necesar suplimentar pentru **acces FTP paw**:
      - **sudo usermod -a -G upload paw**
      - **sudo chmod -R 775 /var/www**

# Project

# Proiect

- Teme in **echipa**: 2/3 membri
- Evaluare **individuala**
- Variabile ca dificultate (cu note diferite)

# PROIECT (final)

- Tema de nota 8
  - Tema unica pentru fiecare student
  - Baza de date cu care se lucreaza contine minim 20 de inregistrari in tabelul cel mai "voluminos«
- Tema de nota 9
  - Conditiiile de la tema de nota 8 **si in plus**
  - Necesitatea conlucrarii intre 2 studenti cu doua teme "pereche"
  - Se accepta ca un student sa realizeze ambele puncte
  - Numar **minim** de pagini dinamice (php+mysql) in aplicatie **4 = 2 X 2**
  - Baza de date cu care se lucreaza contine minim 50 de inregistrari in tabelul cel mai "voluminos"

# PROIECT (final)

- Tema de nota **10**
  - Condițiile de la tema de nota 9 **și în plus**
  - Necesitatea colaborării între 2 studenți cu teme "pereche"
  - Tema se predă/trimite cu macar 1 zi înaintea **sustinerii** ei
  - Numar **minim** de pagini dinamice (php+mysql) în aplicație **6 = 3 X 2**
  - Baza de date cu care se lucrează să conțină minim 100 de înregistrări în tabelul cel mai "voluminos".

# PROIECT (final)

- Tema de nota **10+**
  - Condițiile de la tema de nota 10 **si in plus**
  - Numar **minim** de pagini dinamice (php+mysql) in aplicatie **8 = 4 X 2**
  - Baza de date cu care se lucreaza contine minim **300** de inregistrari in tabelul cel mai "voluminos"
  - Necesitatea investigarii posibilitatilor de **imbunatatire** a aplicatiei si adaugarii de functionalitate (**obligatoriu**)
  - nota individuala la proiect va depinde intr-o mica masura (in limita a 1p) de nota minima a colegilor din echipa
  - **+1p la nota de examen**

# PROIECT (final)

- In caz de necesitate, pentru completarea echipei cadrul didactic poate fi membru al echipelor (9/10/10+). Conditii:
  - metoda de comunicare in echipa sa fie prin email sau direct
  - latentă de raspuns: ~ 1 zi
  - reactiv
  - nota implicita 10 ( 😊 )
  - nu lucreaza noaptea, si in special nu in noaptea dinaintea predarii ( 😊 )
- dezavantaj asumat: "spion" in echipa

# PROIECT (final)

- Tema bonus (>5, in general **offline**)
  - Conditiiile de la tema de nota 10+ **si in plus:**
  - **3 studenti/CD**
  - Baza de date cu care se lucreaza contine minim **500** de inregistrari in tabelul cel mai "voluminos"
  - Numar **minim** de pagini dinamice (php+mysql) in aplicatie **15 = 5 X 3**
  - Tema care face apel la controlul **sesiunii** client/server
  - Necesitatea utilizarii **Javascript** in **aplicatie** (aplicatie libera dar cu efect tehnic nu estetic)
  - Forma paginii controlata dual prin CSS, desktop/phone
  - Facilitati in ceea ce priveste nota (**DACA** toate celelalte conditii sunt indeplinite), la alegere:
    - prezenta la laborator: N → P = **66%**, L = **0%**, E = 33%
    - **+2p la nota de examen**



# PROIECT (final)

- proiectul se **sustine individual** (oral si practic)
- fiecare membru al unei echipe (la temele de nota 9 si 10) trebuie sa sustina in aceeasi zi proiectul
- nota individuala la proiect va depinde intr-o mica masura (in limita a 1p) de nota medie a colegilor din echipa (numai la temele de 10+)
  - $N-\min(E)=1 \rightarrow -0.5 \text{ p}$
  - $N-\min(E)=2 \rightarrow -1 \text{ p}$
  - $N-\min(E)=3 \rightarrow -1.5 \text{ p}$

# Notare proiect 2020/2021

- 1p – functionalitate ✓
- 1p – aplicatia ruleaza pe server-ul CentOS/Ubuntu ✓
- numar de pagini dinamice ✓
- numar de inregistrari in baza de date ✓
- 1p – planul aplicatiei ✓
- 2p – prezentare in Teams a proiectului ✓

# Notare 2016

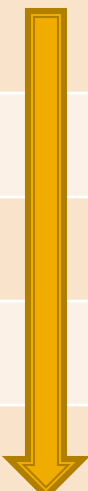
- 1p – functionalitate
- 1p – mutarea **personală** a site-ului (restaurare backup + setare server) pe un server de referință
  - server-ul de referință va fi mașina virtuală **Centos 7.1** utilizată la laborator (inclusiv aplicațiile cu pricina)
  - să vă pregătiți pentru situația în care pe acel server există și alte baze de date care **nu** trebuie distruse
  - fiecare student își pune sursele în directorul propriu, în rădăcina server-ului. Dacă tema depinde de anumite fișiere ale colegului, le cereți înainte
- 1p – cunoașterea codului
  - răspunsul la întrebări de genul: “unde ai făcut aceasta”
- Teme “de nota 10,10+”
  - inițiativă. Investigarea posibilităților de îmbunătățire
  - întrebări legate de cooperarea cu colegul de echipă
  - explicații relativ la funcționarea unei anumite secvențe de cod
  - utilizare sesiune, Javascript, F shape pattern (CSS media)

# Notare 2023 (final)

- 1p – **functionalitate**
- 1p – mutarea **personală** a site-ului (restaurare backup + setare server) pe un server de referință CentOS/**Ubuntu**
- 1p – cunoasterea **codului**
  - răspunsul la întrebări de genul: "unde ai făcut aceasta", "ce face acest cod"
- 1p – **planul aplicației**
- Teme "de nota 10,10+"
  - Inițiativa. Investigarea posibilităților de îmbunătățire
  - Explicații relativ la funcționarea unei anumite secvențe de cod
  - Utilizare sesiune, Javascript, **CSS media**

# Notare 2022

- numar de pagini dinamice ✓
- numar de inregistrari in baza de date ✓
  - se verifica indeplinirea conditiilor corespunzatoare si se realizeaza **de-clasificarea** temei pana cand **ambele** conditii sunt indeplinite

Tema de nota ...		Pagini	Inregistrari
	bonus	$15 = 5 \times 3$	500
	10+	$8 = 4 \times 2$	300
	10	$6 = 3 \times 2$	100
	9	$4 = 2 \times 2$	50
	8	$1 = 1 \times 1$	20

# Exemplu

- 1. Galerie de imagini in care imaginile sunt ordonate dupa categorii.

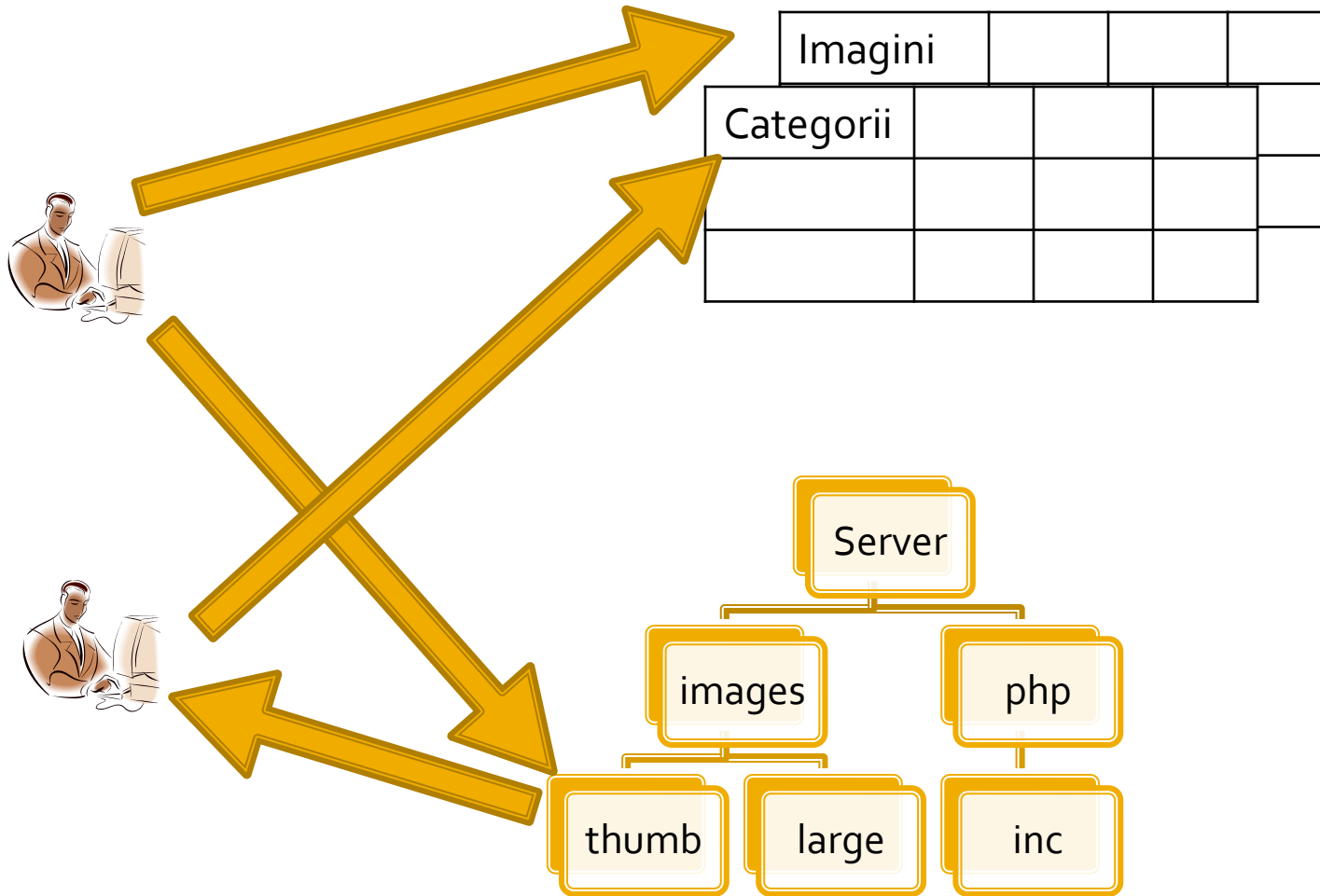


- a. aplicatia pentru adaugarea de categorii si afisare a imaginilor (cu alegerea prealabila a categoriei si afisarea listei de imagini format mic)



- b. aplicatia pentru adaugare de imaginilor (cu alegerea prealabila a categoriei si generarea prealabila a imaginii format mic)

# Exemplu



# “Examen” Alocare teme proiect

- **Alocare teme**

- tema aleasa (optiune principala) - **necesar**
- nume coechipier - **necesar**
- tema alternativa (rezerva 1)
- tema alternativa (rezerva 2)
- punctul ales (a/b) - **necesar**

- Primul venit, primul servit

- **ambii** parteneri finalizeaza examenul



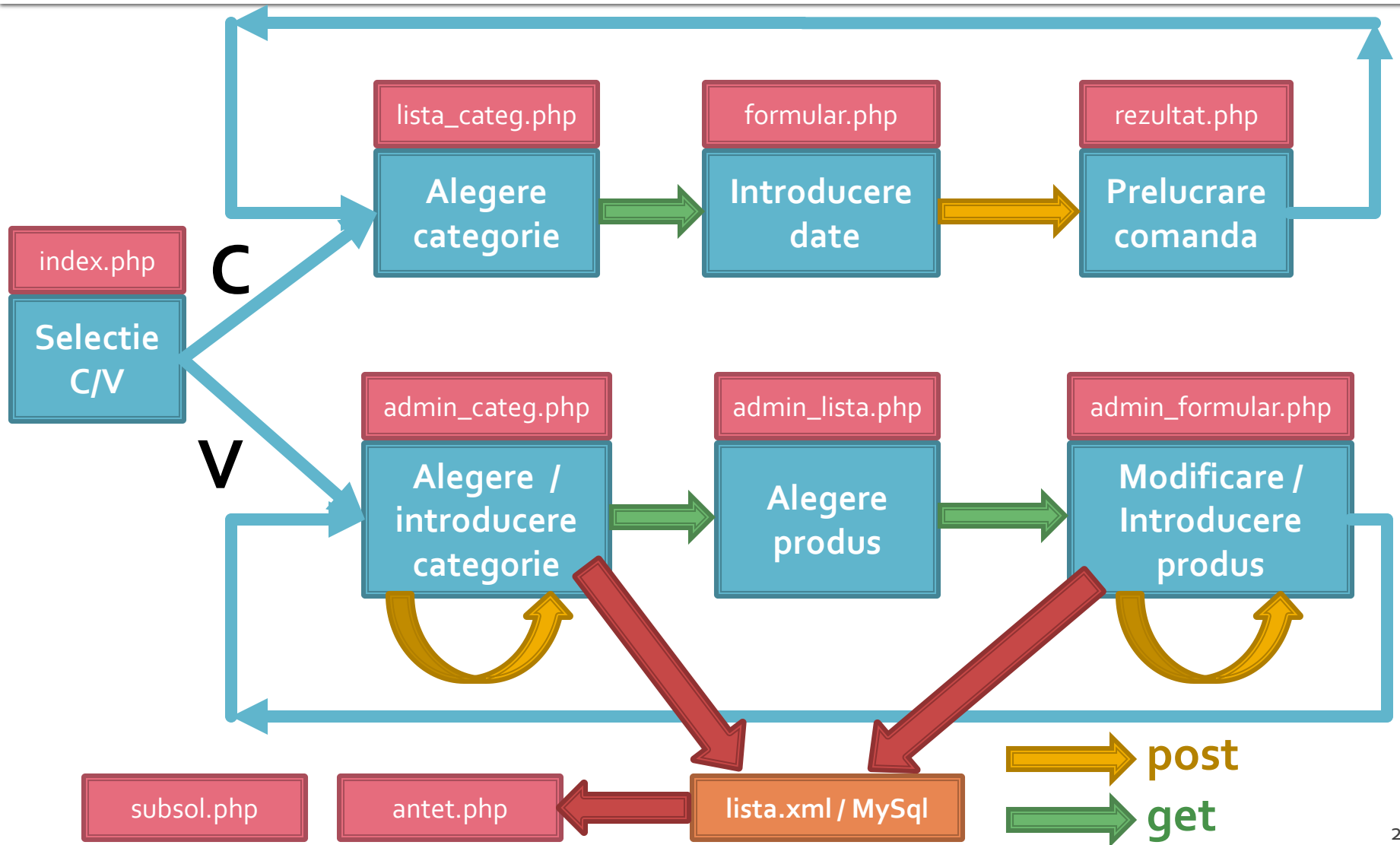
# “Examen” Predare proiect

- Predare proiect
- Chiar daca unele fisiere sunt comune, **ambii** coechipieri trebuie sa finalizeze depunerea
- Predare 3 fisiere
  - un fișier **\*.pdf/\*.jpg** cu **planul aplicației**
  - un fișier **\*.sql** cu backup-ul bazei de date de care are nevoie aplicația pentru a funcționa (**nr. linii!!**)
  - un fișier cu arhiva directorului conținând aplicația (fișiere \*.php, \*.jpg, structură de directoare etc., arhivate: **\*.zip, \*.7z** etc.) (**nr. pagini!!**)

# Server referinta LAMP

- Linux, doua variante
  - Centos 7.1
    - PHP 5.4.16
    - MariaDB 5.5.44 / root:masterrc
    - Apache 2.4.6
    - **root**/student:masterrc
  - **Ubuntu** 20.04
    - PHP 7.4.3
    - MariaDB 10.3.31 / root:masteretti
    - Apache 2.4.41
    - **paw**/student:masteretti
    - necesar suplimentar pentru **acces FTP paw**:
      - **sudo usermod -a -G upload paw**
      - **sudo chmod -R 775 /var/www**

# Plan aplicatie



# Examen

- **fizic**
- probleme
- fiecare student are subiect propriu
- toate materialele permise
- tehnica de calcul **nu** este necesara dar este permisa

# Examen

- Oricare din temele de proiect (sau asemenea) poate constitui una din problemele de examen
  - se va cere realizarea planului / structurii logice a aplicatiei
- Se poate cere scrierea unui cod pentru realizarea anumitor operatii, fara necesitatea corectitudinii tehnice absolute (";", nume corect al functiilor, parametri functie etc.)
- Se poate cere interpretarea unui cod php/MySql cu identificarea efectului

# Aspecte practice recomandate in realizarea aplicatiilor web

# Metode de lucru recomandate 1

- Daca nu aveti acces simplu la “log-urile” server-ului **MySQL** puteti vedea cum ajung efectiv interogariile la el afisand temporar textul interogarii
  - `$query = "SELECT * FROM `produse` AS p  
WHERE `id_categ` = ".$row_result_c['id_categ'];  
echo $query; //util in perioada de testare`
    - Textul prelucrat de PHP al interogarii va fi afisat in clar pe pagina facand mai usoara depanarea programului
    - Aceste linii **trebuie** eliminate in forma finala a programului ca masura de securitate

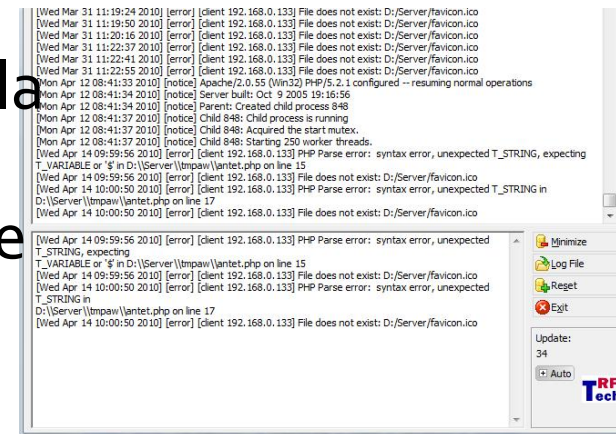
# Metode de lucru recomandate 2

- Verificarea “log-ului” de erori al server-ului Apache ramane principala metoda de depanare a codului PHP.

- W2000: Utilizarea aplicatiei prezentata la laborator este mai comoda datorita automatizarii dar orice alta varianta este utila

- Centos 7.1:

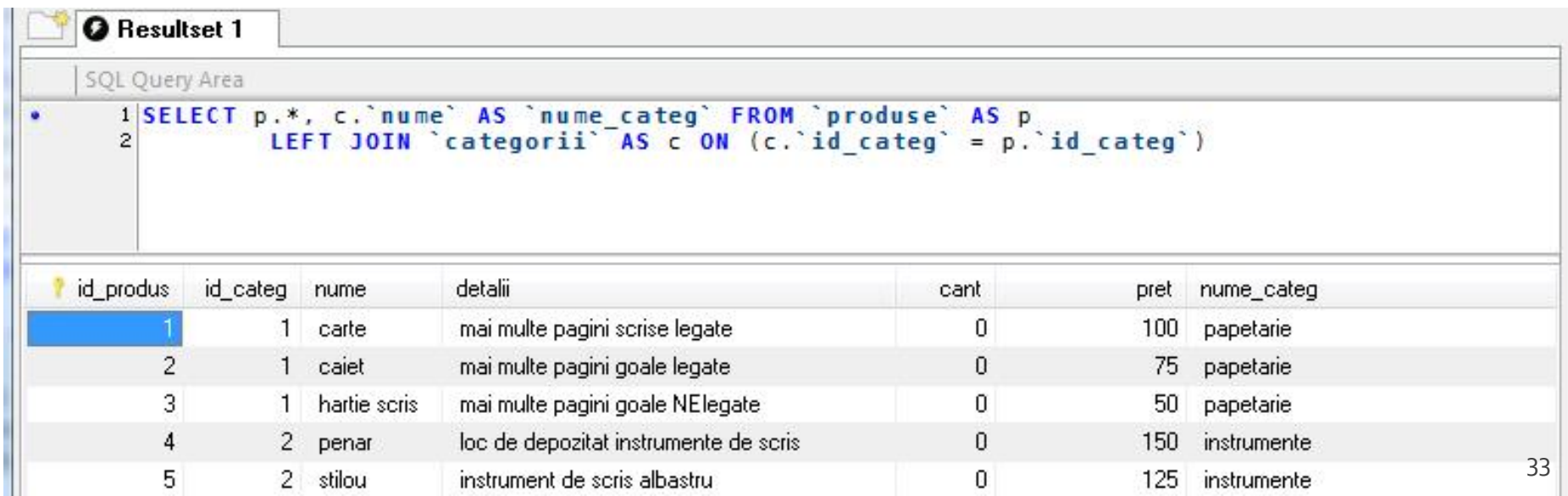
- putty → nano /var/log/httpd/error\_log
- <http://192.168.30.5/logfile.php> (nonstandard)
- tema suplimentara (php.ini + log PHP **recomandat**)





# Metode de lucru recomandate 3

- In perioada de definitivare a formei interogarilor MySql este de multe ori benefic sa se utilizeze mai intai **MySql Workbench/PhpMyAdmin** pentru incercarea interogarilor, urmand ca apoi, cand sunteti multumiti de rezultat, sa transferati interogarea SQL in codul PHP



The screenshot shows the MySQL Workbench interface. At the top, there's a tab labeled 'Resultset 1'. Below it is the 'SQL Query Area' containing the following SQL query:

```
1 SELECT p.*, c.`nume` AS `nume_categ` FROM `produse` AS p
2 LEFT JOIN `categorii` AS c ON (c.`id_categ` = p.`id_categ`)
```

Below the query area, the results are displayed in a table with the following columns: id\_produș, id\_categ, nume, detalii, cant, pret, and nume\_categ. The table contains 5 rows of data.

id_produș	id_categ	nume	detalii	cant	pret	nume_categ
1	1	carte	mai multe pagini scrise legate	0	100	papetarie
2	1	caiet	mai multe pagini goale legate	0	75	papetarie
3	1	hartie scris	mai multe pagini goale NElegate	0	50	papetarie
4	2	penar	loc de depozitat instrumente de scris	0	150	instrumente
5	2	stilou	instrument de scris albastru	0	125	instrumente

# Metode de lucru recomandate 3

MySQL Query Browser - Connection: root@server / tmpaw

File Edit View Query Script Tools Window Help

Transaction Explain Compare

Resultset 1

SQL Query Area

```
1 SELECT p.*, c.`nume` AS `nume_categ` FROM `produse` AS p
2 LEFT JOIN `categorii` AS c ON (c.`id_categ` = p.`id_categ`)
```

id_produs	id_categ	nume	detalii	cant	pret	nume_categ
1	1	carte	mai multe pagini scrise legate	0	100	papetarie
2	1	caiet	mai multe pagini goale legate	0	75	papetarie
3	1	hartie scris	mai multe pagini goale NElegate	0	50	papetarie
4	2	penar	loc de depozitat instrumente de scris	0	150	instrumente
5	2	stilou	instrument de scris albastru	0	125	instrumente
6	2	creion	instrument de scris gri	0	25	instrumente
7	3	cd	canta	0	50	audio-video
8	3	dvd	vizual	0	100	audio-video
9	3	blue ray	vizual extrem	0	500	audio-video

9 rows fetched in 0.0035s (0.0016s)

Edit Apply Changes Discard Changes First Last Search

1: 1

# Metode de lucru recomandate 4

- eficienta unei aplicatii web
  - 100% - toate prelucrarile "mutate" in RDBMS
  - PHP doar afisarea datelor
- eficienta unei aplicatii MySql
  - 25% alegerea corecta a tipurilor de date
  - 25% crearea indecsilor necesari in aplicatii
  - 25% normalizarea corecta a bazei de date
  - 20% cresterea complexitatii interogarilor pentru a "muta" prelucrarile pe server-ul de baze de date
  - 5% scrierea corecta a interogarilor

# Metode de lucru recomandate 5a

- La implementarea unei aplicatii noi (proiect)
  1. Imaginarea planului aplicatiei (ex: S25)
    - "cum as vrea eu sa lucrez cu o astfel de aplicatie"
    - hartie/creion/timp – esentiale
  2. Identificarea datelor/transmisia de date intre pagini
    - get/post/fisier unic colectare-prelucrare
    - baza de date read/write
  3. Identificarea structurii logice a datelor utilizate
    - "clase" de obiecte/fenomene tratate identic
    - se are in vedere scalabilitatea (posibilitatea de crestere a numarului de elemente dintr-o clasa)

# Metode de lucru recomandate 5b

- La implementarea unei aplicatii noi (proiect)
  - 4. Realizarea structurii bazei de date
    - In general un tabel pentru fiecare clasa logica distincta **DAR...**
    - se are in vedere scalabilitatea (daca aplicatia creste sa **NU** apara cresterea numarului de clase/tabele) **SI...**
    - normalizare
  - 5. Identificarea tipului de date necesar pentru coloane
    - de preferat numerele intregi in orice situatie care presupune ordonare
    - dimensiunea campurilor nu mai mare decat e necesar (poate fi fortata prin atributul "size" in eticheta HTML "input")
  - 6. Imaginarea forme fizice a paginilor
    - "am mai vazut asa si mi-a placut" (Don't make me think!)
    - investigarea posibilitatii de a introduce functionalitate template

# Metode de lucru recomandate 5c

- La implementarea unei aplicatii noi (proiect)
  - 7. Popularea manuala a bazei de date cu date initiale
    - MySql Query Browser (sau PhpMyAdmin) / automat / imprumut
    - programarea individuala a paginilor are nevoie de prezenta unor date
  - 8. Programare individuala a paginilor
    - In general in ordinea din planul aplicatiei (de multe ori o pagina asigura datele necesare pentru urmatoarea din plan)
    - modul "verbose" activ pentru PHP (adica: `echo $a; print_r($matr)`)
  - 9. Pregatirea pentru distributie/mutare
    - testare detaliata (eventual un "cobai")
    - eliminarea adaosurilor "verbose"
    - backup
    - generarea unui eventual install/setup

# Lim baj SQL

# Referinta relativa

- Referinta la elementele unei baze de date se face prin utilizarea numelui elementului respectiv daca nu exista dubii (referinta relativa)
  - daca baza de date este selectata se poate utiliza numele tabelului pentru a identifica un tabel
    - `USE db_name;`  
`SELECT * FROM tbl_name;`
  - daca tabelul este identificat in instructiune se poate utiliza numele coloanei pentru a identifica coloana implicata
    - `SELECT col_name FROM tbl_name;`



# Referinta absoluta

- In cazul in care apare ambiguitate in identificarea unui element se poate indica descendenta sa pâna la disparitia ambiguitatii
- Astfel, o anumita coloana, `col_name`, care apartine tabelului `tbl_name` din baza de date (schema) `db_name` poate fi identificata in functie de necesitati ca:
  - `col_name`
  - `tbl_name.col_name`
  - `db_name.tbl_name.col_name`

# Nume de identificatori permise

- Numele de identificatori pot avea o lungime de reprezentare de maxim 64 octeti cu exceptia Alias care poate avea o lungime de 255 octeti
- Nu sunt permise:
  - caracterul NULL (ASCII 0x00) sau 255 (0xFF)
  - caracterul "/"
  - caracterul "\"
  - caracterul "."
- Numele nu se pot termina cu caracterul spatiu

# Nume de identificatori permise

- Numele de baze de date nu pot contine decat caractere permise in numele de directoare
- Numele de tabele nu pot contine decat caractere permise in numele de fisiere
- Anumite caractere utilizate vor impune necesitatea trecerii intre apostroafe a numelui
- Apostroful utilizat pentru nume de identificatori e apostroful invers (**backtick**) “`”
  - pentru a nu aparea confuzie cu variabilele sir
  - nu necesita aparitia apostrofului caracterele alfanumerice normale, “\_”, “\$”
- numele rezervate trebuie de asemenea cuprinse intre apostroafe pentru a fi utilizate

# Alias

- Orice identificator poate primi un nume asociat
  - **Alias**
    - pentru a elimina ambiguitati
    - pentru a usura scrierea
    - pentru a modifica numele coloanelor in rezultate
- Definirea unui alias se face in interiorul unei interogari SQL si are efect in aceeasi interogare
  - `SELECT `t`.* FROM `tbl_name` AS t;`
  - `SELECT `t`.* FROM `tbl_name` t;`

# Alias

- Desi utilizarea cuvintului cheie AS nu este obligatorie, obisnuinta utilizarii lui este recomandata, pentru a evita/identifica alocari eronate
  - `SELECT id, nume FROM produse;` ← doua coloane
  - `SELECT id nume FROM produse;` ← Alias "nume" creat pentru coloana "id"

# Alias

- Usurinta scrierii
  - `SELECT * FROM un_tabel_cu_numelung AS t WHERE t.col1 = 5 AND t.col2 = 'ceva'`
- Modificarea numelui de coloana, sau crearea unui nume pentru o coloana calculata in rezultate
  - `SELECT CONCAT(nume,' ',prenume) AS nume_intreg FROM studenti AS s;`
  - `SELECT `n1` AS `Nume`, `n2` AS `Nota`, `n3` AS `Numar matricol` FROM elevi AS e;`

# Alias

- Eliminarea ambiguitatilor
  - intalnita frecvent la relatii "many to many"
  - ```
SELECT p.*, c.`nume` AS `nume_categ` FROM  
`produse` AS p  
LEFT JOIN `categorii` AS c ON (c.`id_categ` =  
p.`id_categ`);
```
  - tabelele c si p contin ambele coloanele "nume" si "id\_categ"
    - modificarea denumirii coloanei "nume" din categorii pentru evitarea confuziei cu coloana "nume" din produse
    - eventual se pot da nume diferite coloanelor "id\_categ" pentru a evita ambiguitatea in interiorul clauzei ON (desi si referinta absoluta rezolva aceasta problema)

# Interrogari SQL



# Interogari

- Interogările SQL pot fi
  - Pentru definirea datelor, crearea programatica de baze de date, tabele, coloane etc.
    - mai puțin utilizate în majoritatea aplicațiilor
    - ALTER, CREATE, DROP, RENAME
  - Pentru manipularea datelor
    - SELECT, INSERT, UPDATE, REPLACE etc.
  - Pentru control/administrare tranzacții/server
- De cele mai multe ori aplicațiile doar manipulează datele. Structura este definită în avans de asemenea și administrarea este mai facilă cu programe specializate
- Următoarele definiții sunt cele valabile pentru **MySQL 5.0**

# ALTER DATABASE

- ALTER {DATABASE | SCHEMA} [db\_name] alter\_specification ...
  - alter\_specification:
    - [DEFAULT] CHARACTER SET [=] charset\_name
    - [DEFAULT] COLLATE [=] collation\_name
- Modifica caracteristicile generale ale unei baze de date
- E necesar dreptul de acces (privilegiu) ALTER asupra respectivei baze de date

# ALTER TABLE

- ALTER TABLE {table\_option [, table\_option] ... | partitioning\_specification}
  - table\_option:
    - ADD [COLUMN] col\_name column\_definition [FIRST | AFTER col\_name ]
    - ADD {INDEX|KEY} [index\_name] [index\_type] (index\_col\_name,...) [index\_option] ...
    - ADD [CONSTRAINT [symbol]] PRIMARY KEY [index\_type] (index\_col\_name,...) [index\_option]
    - ...
    - CHANGE [COLUMN] old\_col\_name new\_col\_name column\_definition [FIRST|AFTER col\_name]
    - MODIFY [COLUMN] col\_name column\_definition [FIRST | AFTER col\_name]
    - DROP [COLUMN] col\_name
    - DROP PRIMARY KEY
    - DROP {INDEX|KEY} index\_name
    - DISABLE KEYS
    - ENABLE KEYS
    - RENAME [TO] new\_tbl\_name
- permite modificarea unui tabel existent

# CREATE DATABASE

- `CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name [create_specification...]`
  - `create_specification:`
    - `[DEFAULT] CHARACTER SET charset_name`
    - `[DEFAULT] COLLATE collation_name`
- Crearea unei noi baze de date
- Necesara la instalarea unei aplicatii
- Fisierile SQL "backup" contin succesiunea `DROP...`, `CREATE...` pentru a inlocui datele in intregime

# CREATE INDEX

- CREATE [UNIQUE|FULLTEXT|SPATIAL]  
INDEX index\_name [USING index\_type] ON  
tbl\_name (index\_col\_name,...)
  - index\_col\_name:
    - col\_name [(length)] [ASC | DESC]
- Crearea unui index se face de obicei la crearea tabelului
- Interogarea CREATE INDEX ... se transpune in interogare ALTER TABLE ...

# CREATE TABLE

- `CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_definition,...)] [table_options] [select_statement]`
- `CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [( ) LIKE old_tbl_name ( )]`
- Interogarea de creare a tabelului este memorata intern de server-ul MySql pentru utilizari ulterioare (in general in `ALTER TABLE` sa fie cunoscute specificatiile initiale)

# CREATE TABLE

- create\_definition – coloana impreuna cu eventualele caracteristici (in special chei - indecsi):
  - column\_definition
    - | [CONSTRAINT [symbol]] PRIMARY KEY [index\_type] (index\_col\_name,...)
    - | KEY [index\_name] [index\_type] (index\_col\_name,...)
    - | INDEX [index\_name] [index\_type] (index\_col\_name,...)
    - | [CONSTRAINT [symbol]] UNIQUE [INDEX] [index\_name] [index\_type] (index\_col\_name,...)
    - | [FULLTEXT|SPATIAL] [INDEX] [index\_name] (index\_col\_name,...)
    - | [CONSTRAINT [symbol]] FOREIGN KEY [index\_name] (index\_col\_name,...) [reference\_definition]
    - | CHECK (expr)
- column\_definition – nume si tipul de date (curs 7-8):
  - col\_name type [NOT NULL | NULL] [DEFAULT default\_value] [AUTO\_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY] [COMMENT 'string'] [reference\_definition]

# CREATE TABLE

- Exemple

- CREATE TABLE test (a INT NOT NULL AUTO\_INCREMENT, PRIMARY KEY (a), KEY(b)) SELECT b,c FROM test2;
- CREATE TABLE IF NOT EXISTS `schema`.`Employee` (  
`idEmployee` VARCHAR(45) NOT NULL,  
`Name` VARCHAR(255) NULL,  
`idAddresses` VARCHAR(45) NULL,  
PRIMARY KEY (`idEmployee`),  
CONSTRAINT `fkEmployee\_Addresses`  
FOREIGN KEY `fkEmployee\_Addresses` (`idAddresses`)  
REFERENCES `schema`.`Addresses` (`idAddresses`)  
ON DELETE NO ACTION  
ON UPDATE NO ACTION)  
ENGINE = InnoDB  
DEFAULT CHARACTER SET = utf8  
COLLATE = utf8\_bin



# CREATE TABLE

- `CREATE ... LIKE ...` creaza un tabel fara date pe baza modelului unui tabel existent. Se pastreaza definitiile coloanelor si eventualele chei (index) definite in tabelul anterior
- `CREATE ... SELECT ...` creaza un tabel cu date pe baza modelului si datelor obtinute dintr-un alt tabel existent. Sunt obtinute anumite coloane (`SELECT`) cu tipul lor, dar fara crearea indecsilor
- `CREATE TEMPORARY TABLE` creaza un tabel temporar. Utilizat in cazul interogarilor complexe sau cu numar mare de rezultate

# DROP

- `DROP {DATABASE | SCHEMA} [IF EXISTS]`  
`db_name`
- `DROP INDEX index_name ON tbl_name`
- `DROP [TEMPORARY] TABLE [IF EXISTS]`  
`tbl_name [, tbl_name] ...`
- Trebuie utilizate cu foarte mare atentie aceste interogari, stergerea datelor este ireversibila
- Fisierile SQL "backup" contin succesiunea `DROP...`, `CREATE...` pentru a inlocui datele in intregime

# Interrogari SQL

# Interogari

- Interogările SQL pot fi
  - Pentru definirea datelor, crearea programatica de baze de date, tabele, coloane etc.
    - mai putin utilizate in majoritatea aplicatiilor
    - ALTER, CREATE, DROP, RENAME
  - **Pentru manipularea datelor**
    - SELECT, INSERT, UPDATE, REPLACE, DELETE etc.
  - Pentru control/administrare tranzactii/server
- De cele mai multe ori aplicatiile doar manipuleaza datele. Structura este definita in avans de asemenea si administrarea este mai facila cu programe specializate
- Urmatoarele definitii sunt cele valabile pentru **MySql 5.0**

# DELETE

- `DELETE [LOW_PRIORITY] [QUICK] [IGNORE]`  
`FROM table_name [WHERE where_condition]`  
`[ORDER BY ...] [LIMIT row_count]`
- Sterge linii din tabelul mentionat si returneaza  
numarul de linii sterse
- `[LOW_PRIORITY] [QUICK] [IGNORE]` sunt  
optiuni care instruiesc server-ul sa reactioneze  
diferit de varianta standard
- Exemplu:
  - `DELETE FROM somelog WHERE user = 'jcole'`  
`ORDER BY timestamp_column LIMIT 1;`

# DELETE

- [WHERE where\_condition] – folosit pentru a selecta liniile care trebuie sterse
  - In absenta conditiei se sterg **toate liniile** din tabel
- [LIMIT row\_count] sterge numai *row\_count* linii dupa care se opreste
  - In general pentru a limita ocuparea server-ului (recrearea indecsilor se face “on the fly”)
  - Operatia se poate repeta pana valoarea returnata e mai mica decat row\_count
- [ORDER BY ...] precizeaza ordinea in care se sterg liniile identificate prin conditie

# INSERT

- INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE] [INTO] tbl\_name [(col\_name,...)] VALUES ({expr | DEFAULT},...),(...),... [ON DUPLICATE KEY UPDATE col\_name=expr, ... ]
- INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE] [INTO] tbl\_name SET col\_name={expr | DEFAULT}, ... [ON DUPLICATE KEY UPDATE col\_name=expr, ... ]
- INSERT [LOW\_PRIORITY | HIGH\_PRIORITY] [IGNORE] [INTO] tbl\_name [(col\_name,...)] SELECT ... [ ON DUPLICATE KEY UPDATE col\_name=expr, ... ]

# INSERT

- Introduce linii noi intr-un tabel
- Primele doua forme introduc valori exprimate explicit
  - INSERT ... VALUES ...
  - INSERT ... SET ...
- INSERT ... SELECT ... introduce valori rezultate obtinute printr-o interogare SQL
- DELAYED – interogarea primeste raspuns de la server imediat, dar inserarea datelor se face efectiv cand tabelul implicat nu este folosit
  - valabil pentru metodele de stocare MyISAM, Memory, Archive



# INSERT

## ■ Exemple

- `INSERT INTO tbl_name (a,b,c) VALUES (1,2,3), (4,5,6), (7,8,9);`
- `INSERT INTO tbl_name (col1,col2) VALUES (15,col1*2);`
- `INSERT INTO table1 (field1,field3,field9) SELECT field3,field1,field4 FROM table2;`

# INSERT

- INSERT ... ON DUPLICATE KEY UPDATE ...
- Daca inserarea unei noi linii ar conduce la duplicarea unei chei primare sau unice, in loc sa se introduca o noua linie se modifica linia anterioara
- Exemple
  - INSERT INTO table (a,b,c) VALUES (1,2,3) ON DUPLICATE KEY UPDATE c=c+1;
  - INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6) ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);

# REPLACE

- REPLACE [LOW\_PRIORITY | DELAYED] [INTO] tbl\_name [(col\_name,...)] VALUES ({expr | DEFAULT},...),(...),...
- REPLACE [LOW\_PRIORITY | DELAYED] [INTO] tbl\_name SET col\_name={expr | DEFAULT}, ...
- REPLACE [LOW\_PRIORITY | DELAYED] [INTO] tbl\_name [(col\_name,...)] SELECT ...
- REPLACE functioneaza similar cu INSERT
  - daca noua linie nu realizeaza duplicarea unei chei primare sau unice se realizeaza insertie
  - daca noua linie realizeaza duplicarea unei chei primare sau unice se sterge linia anterioara dupa care se insereaza noua linie
- REPLACE e extensie MySql a limbajului SQL standard

# UPDATE

- UPDATE [LOW\_PRIORITY] [IGNORE] tbl\_name SET col\_name1=expr1 [, col\_name2=expr2 ...] [WHERE where\_condition] [ORDER BY ...] [LIMIT row\_count]
- Modificarea valorilor stocate intr-o linie
- Exemple
  - UPDATE persondata SET age=15 WHERE id=6;
  - UPDATE persondata SET age=age+1;

# SELECT

- SELECT [ALL | DISTINCT | DISTINCTROW ]  
[HIGH\_PRIORITY] [STRAIGHT\_JOIN]  
select\_expr, ... [FROM table\_references  
  - [WHERE where\_condition]
  - [GROUP BY {col\_name | expr | position} [ASC | DESC],  
... [WITH ROLLUP]]
  - [HAVING where\_condition]
  - [ORDER BY {col\_name | expr | position} [ASC | DESC],  
...]
  - [LIMIT {[offset,] row\_count | row\_count OFFSET  
offset}]
- ]

# SELECT

- SELECT este **cea mai importanta** interogare SQL.
- Intelegerea setarilor si utilizarea inteligenta a indecsilor stau la baza eficientei unei aplicatii
- E absolut necesara realizarea interogarii in asa fel incat datele returnate sa fie exact cele dorite (prelucrarea sa se realizeze pe server-ul MySql)

# SELECT

- `select_expr`: macar o expresie selectata trebuie sa apara
  - identifica ceea ce trebuie extras ca valori de iesire din baza de date
  - pot fi nume de coloana(e)
  - pot fi date de sinteza (rezultate din utilizarea unor functii MySql) – necesara atribuirea unui Alias
    - `SELECT CONCAT(last_name,', ',first_name) AS full_name FROM mytable ORDER BY full_name;`

# SELECT

- WHERE where\_condition, HAVING where\_condition sunt utilizate pentru a introduce criterii de selectie
  - in general au comportare similara si sunt interschimbabile
  - WHERE accepta orice operatori mai putin functii aggregate – de “sumare” (COUNT, MAX)
  - HAVING accepta functii aggregate, dar se aplica la sfarsit, exact inainte de a fi trimise datele clientului, **fara nici o optimizare** – utilizarea este recomandata doar cand nu exista echivalent WHERE



# SELECT

- ORDER BY {col\_name | expr | position} [ASC | DESC]
  - ordoneaza datele returnate dupa anumite criterii (valoarea unei anumite coloane sau functii).
    - Implicit ordonarea este crescatoare ASC, dar se poate specifica ordine descrescatoare DESC
- GROUP BY {col\_name | expr | position}
  - realizeaza gruparea liniilor returnate dupa anumite criterii
  - permite utilizarea functiilor aggregate (de sumare)

# SELECT

- GROUP BY – functii aggregate
  - AVG(expresie) – mediere valorilor
    - SELECT student\_name, AVG(test\_score) FROM student GROUP BY student\_name;
  - COUNT(expresie), COUNT(\*)
    - SELECT COUNT(\*) FROM student;
    - SELECT COUNT(DISTINCT results) FROM student;
    - SELECT student.student\_name, COUNT(\*) FROM student, course WHERE student.student\_id=course.student\_id GROUP BY student\_name;
    - SELECT columnname, COUNT(columnname) FROM tablename GROUP BY columnname HAVING COUNT(columnname)>1
- Cuvantul cheie DISTINCT este utilizat pentru a procesa doar liniile cu valori diferite
  - exemplu: 100 de note (rezultate) la examen
    - COUNT(results) va oferi raspunsul 100
    - COUNT(DISTINCT results) va oferi raspunsul 7 (notele diferite 4,5,6,7,8,9,10)

# SELECT

- GROUP BY – functii aggregate
  - MIN(expresie), MAX(expresie) – minim si maxim
    - SELECT student\_name, MIN(test\_score), MAX(test\_score) FROM student GROUP BY student\_name;
  - SUM(expresie) – sumarea valorilor
    - SELECT year, SUM(profit) FROM sales GROUP BY year;
- WITH ROLLUP – operatii de sumare super-aggregate (un nivel suplimentar de agregare)

# SELECT ... WITH ROLLUP

- `SELECT year, SUM(profit) FROM sales GROUP BY year;`
- `SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;`
  - se obtine un total general, linia "super-aggregate" este identificata dupa valoarea NULL a coloanei dupa care se face sumarea

| year | SUM(profit) |
|------|-------------|
| 2000 | 4525        |
| 2001 | 3010        |

| year | SUM(profit) |
|------|-------------|
| 2000 | 4525        |
| 2001 | 3010        |
| NULL | 7535        |

# SELECT

- LIMIT [offset,] row\_count | row\_count
  - se limiteaza numarul de linii returnate
  - utilizat **frecvent** in aplicatiile web
  - LIMIT 15 – returneaza doar primele 15 linii (1÷15)
  - LIMIT 10,15 – returneaza 15 linii dupa primele 10 linii (11÷25)

# JOIN

- Normalizarea si existenta relatiilor intre diversele tabele ale unei baze de date implica faptul ca pentru aflarea unor informatii utilizabile (complete), acestea trebuie extrase **simultan** din mai multe tabele
  - informatie inutilizabila: studentul cu id-ul 253 a luat nota 8 la examenul cu id-ul 35
- Uneori asamblarea informatiilor din mai multe tabele e necesara pentru obtinerea unor rapoarte complexe
  - Exemplu: tabel cu clienti, tabel cu comenzi, tabel cu produse; legatura produse-comenzi e implementata printr-un tabel suplimentar. Raspunsul la intrebarea cate produse x a cumparat clientul y cere tratarea unitara a celor 4 tabele implicate

# JOIN

- In general in SQL se poate descrie o astfel de unificare de date intre doua tabele:
  - `left_table JOIN_type right_table criteriu_unificare`
- JOIN\_type
  - JOIN – selecteaza toate liniile compuse in care criteriul este indeplinit pentru ambele tabele
  - LEFT JOIN – compune si selecteaza toate liniile din `left_table` chiar daca nu este gasit un corespondent in `right_table`
  - RIGHT JOIN – compune si selecteaza toate liniile din `right table` (similar)
  - FULL JOIN – compune si selecteaza toate liniile din `left_table` si `right_table` fie ca este indeplinit criteriul fie ca nu (nu este implementat in MySql, poate fi simulat)

# JOIN

- Clauza JOIN e utilizata pentru a realiza o unificare temporara, dupa anumite criterii, din punct de vedere logic, a doua tabele in vederea extragerii informatiei "suma" dorite
  - left\_table [INNER | CROSS] JOIN right\_table [join\_condition]
  - left\_table STRAIGHT\_JOIN right\_table
  - left\_table STRAIGHT\_JOIN right\_table ON condition
  - left\_table LEFT [OUTER] JOIN right\_table join\_condition
  - left\_table NATURAL [LEFT [OUTER]] JOIN right\_table
  - left\_table RIGHT [OUTER] JOIN right\_table join\_condition
  - left\_table NATURAL [RIGHT [OUTER]] JOIN right\_table
  - join\_condition: ON conditional\_expr | USING (column\_list)



# JOIN – Exemplu

- Tabel clienti
  - 4 clienti
- Tabel comenzi
  - client 1 – 2 comenzi
  - client 2 – 0 comenzi
  - client 3,4 – 1 comanda

```
CREATE TABLE `clienti` (  
  `id_client` int(10) unsigned NOT NULL auto_increment,  
  `nume` varchar(100) NOT NULL,  
  PRIMARY KEY (`id_client`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
INSERT INTO `clienti` (`id_client`,`nume`) VALUES  
(1,'Ionescu'),  
(2,'Popescu'),  
(3,'Vasilescu'),  
(4,'Georgescu');
```

```
CREATE TABLE `comenzi` (  
  `id_comanda` int(10) unsigned NOT NULL auto_increment,  
  `id_client` int(10) unsigned NOT NULL,  
  `suma` double NOT NULL,  
  PRIMARY KEY (`id_comanda`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
INSERT INTO `comenzi` (`id_comanda`,`id_client`,`suma`) VALUES  
(1,1,19.99),  
(2,1,35.15),  
(3,3,17.56),  
(4,4,12.34);
```

# INNER JOIN

- INNER JOIN sunt unificarile implicite, in care criteriul (join\_condition) trebuie indeplinit in ambele tabele (extensie a cuvintului cheie JOIN pentru evitarea ambiguitatii)
  - OUTER JOIN = {LEFT JOIN | RIGHT JOIN | FULL JOIN} – nu e obligatoriu sa fie indeplinit criteriul in ambele tabele
  - FULL JOIN nu e implementat in MySql, poate fi simulat ca UNION intre LEFT JOIN si RIGHT JOIN
- INNER JOIN sunt echivalente cu realizarea produsului cartezian intre cele doua tabele implicate urmata de verificarea criteriului, daca acesta exista

# CROSS JOIN

- In MySql INNER JOIN si CROSS JOIN sunt echivalente in totalitate
  - In SQL standard INNER este folosit in prezenta unui criteriu, CROSS in absenta sa
- INNER (CROSS) JOIN si “,” sunt echivalente cu produsul cartezian intre cele doua tabele implicate in conditiile lipsei criteriului de selectie: fiecare linie a unui tabel este alaturata fiecarei linii din al doilea tabel
  - (un tabel cu M linii si A coloane) CROSS JOIN (un tabel cu N linii si B coloane) → (un tabel cu  $M \times N$  linii si  $A+B$  coloane)

# CROSS JOIN

## SQL Query Area

```
1 SELECT * FROM clienti JOIN comenzi;  
2 SELECT * FROM clienti, comenzi;  
3 SELECT * FROM clienti INNER JOIN comenzi;  
4 SELECT * FROM clienti CROSS JOIN comenzi;
```

| id_client | nume      | id_comanda | id_client | suma  |
|-----------|-----------|------------|-----------|-------|
| 1         | Ionescu   | 1          | 1         | 19.99 |
| 2         | Popescu   | 1          | 1         | 19.99 |
| 3         | Vasilescu | 1          | 1         | 19.99 |
| 4         | Georgescu | 1          | 1         | 19.99 |
| 1         | Ionescu   | 2          | 1         | 35.15 |
| 2         | Popescu   | 2          | 1         | 35.15 |
| 3         | Vasilescu | 2          | 1         | 35.15 |
| 4         | Georgescu | 2          | 1         | 35.15 |
| 1         | Ionescu   | 3          | 3         | 17.56 |
| 2         | Popescu   | 3          | 3         | 17.56 |
| 3         | Vasilescu | 3          | 3         | 17.56 |
| 4         | Georgescu | 3          | 3         | 17.56 |
| 1         | Ionescu   | 4          | 4         | 12.34 |
| 2         | Popescu   | 4          | 4         | 12.34 |
| 3         | Vasilescu | 4          | 4         | 12.34 |
| 4         | Georgescu | 4          | 4         | 12.34 |

# INNER JOIN – criterii

- USING – trebuie sa aiba o coloana cu nume identic in cele doua tabele
  - coloana comuna este afisata o singura data
- ON – accepta orice conditie conditionala
  - chiar daca numele coloanelor din conditie sunt identice, sunt tratate ca entitati diferite (id\_client apare de doua ori provenind din cele doua tabele)

| SQL Query Area |                                                             |            |       |  |
|----------------|-------------------------------------------------------------|------------|-------|--|
| 1              | SELECT * FROM clienti INNER JOIN comenzi USING (id_client); |            |       |  |
| id_client      | nume                                                        | id_comanda | suma  |  |
| 1              | Ionescu                                                     | 1          | 19.99 |  |
| 1              | Ionescu                                                     | 2          | 35.15 |  |
| 3              | Vasilescu                                                   | 3          | 17.56 |  |
| 4              | Georgescu                                                   | 4          | 12.34 |  |

| 1         | SELECT * FROM clienti INNER JOIN comenzi ON (clienti.id_client=comenzi.id_client); |            |           |       |
|-----------|------------------------------------------------------------------------------------|------------|-----------|-------|
| id_client | nume                                                                               | id_comanda | id_client | suma  |
| 1         | Ionescu                                                                            | 1          | 1         | 19.99 |
| 1         | Ionescu                                                                            | 2          | 1         | 35.15 |
| 3         | Vasilescu                                                                          | 3          | 3         | 17.56 |
| 4         | Georgescu                                                                          | 4          | 4         | 12.34 |

# NATURAL JOIN

- NATURAL JOIN e echivalent cu o unificare INNER JOIN cu o clauza USING(...) care utilizeaza toate coloanele cu nume comun intre cele doua tabele

| SQL Query Area |                                                          |           |            |       |
|----------------|----------------------------------------------------------|-----------|------------|-------|
| 1              | <code>SELECT * FROM clienti NATURAL JOIN comenzi;</code> |           |            |       |
| ?              | id_client                                                | nume      | id_comanda | suma  |
|                | 1                                                        | Ionescu   | 1          | 19.99 |
|                | 1                                                        | Ionescu   | 2          | 35.15 |
|                | 3                                                        | Vasilescu | 3          | 17.56 |
|                | 4                                                        | Georgescu | 4          | 12.34 |

# LEFT JOIN

- Unificare de tip OUTER JOIN
- Se returneaza linia din left\_table chiar daca nu exista corespondent in right\_table (se introduc valori NULL)
- Cuvantul cheie OUTER este optional

| SQL Query Area |                                                                 |            |       |
|----------------|-----------------------------------------------------------------|------------|-------|
| 1              | SELECT * FROM clienti LEFT OUTER JOIN comenzi USING(id_client); |            |       |
| id_client      | nume                                                            | id_comanda | suma  |
| 1              | Ionescu                                                         | 1          | 19.99 |
| 1              | Ionescu                                                         | 2          | 35.15 |
| 2              | Popescu                                                         | NULL       | NULL  |
| 3              | Vasilescu                                                       | 3          | 17.56 |
| 4              | Georgescu                                                       | 4          | 12.34 |

# RIGHT JOIN

- Unificare de tip OUTER JOIN
- Se returneaza linia din right\_table chiar daca nu exista corespondent in left\_table
- Echivalent cu LEFT JOIN cu tabelele scrise in ordine inversa

| SQL Query Area |                                                                  |       |           |
|----------------|------------------------------------------------------------------|-------|-----------|
| 1              | SELECT * FROM clienti RIGHT OUTER JOIN comenzi USING(id_client); |       |           |
| id_client      | id_comanda                                                       | suma  | nume      |
| 1              | 1                                                                | 19.99 | Ionescu   |
| 1              | 2                                                                | 35.15 | Ionescu   |
| 3              | 3                                                                | 17.56 | Vasilescu |
| 4              | 4                                                                | 12.34 | Georgescu |

| SQL Query Area |                                                                  |            |       |
|----------------|------------------------------------------------------------------|------------|-------|
| 1              | SELECT * FROM comenzi RIGHT OUTER JOIN clienti USING(id_client); |            |       |
| id_client      | nume                                                             | id_comanda | suma  |
| 1              | Ionescu                                                          | 1          | 19.99 |
| 1              | Ionescu                                                          | 2          | 35.15 |
| 2              | Popescu                                                          | NULL       | NULL  |
| 3              | Vasilescu                                                        | 3          | 17.56 |
| 4              | Georgescu                                                        | 4          | 12.34 |



# JOIN

- `STRAIGHT_JOIN` – forteaza citirea mai intai a valorilor din `left_table` si apoi a celor din `right_table` (in anumite cazuri citirea se realizeaza invers)
- `USE_INDEX`, `IGNORE_INDEX`, `FORCE_INDEX` controlul index-ului utilizat pentru gasirea si selectia liniilor, poate aduce spor de viteza

# UNION

- Combina rezultatele mai multor interogari SELECT intr-un singur rezultat general
- SELECT ... UNION [ALL | DISTINCT]  
SELECT ... [UNION [ALL | DISTINCT]  
SELECT ...]
- Poate fi folosit pentru a realiza FULL JOIN

| SQL Query Area |        |                   |      |                                                                     |
|----------------|--------|-------------------|------|---------------------------------------------------------------------|
| 1              | SELECT | *                 | FROM | comenzi LEFT JOIN clienti ON (comenzi.id_client=clienti.id_client)  |
| 2              | UNION  |                   |      |                                                                     |
| 3              | SELECT | *                 | FROM | comenzi RIGHT JOIN clienti ON (comenzi.id_client=clienti.id_client) |
| 4              | WHERE  | comenzi.id_client | IS   | NULL                                                                |

| id_comanda | id_client | suma  | id_client | nume      |
|------------|-----------|-------|-----------|-----------|
| 1          | 1         | 19.99 | 1         | Ionescu   |
| 2          | 1         | 35.15 | 1         | Ionescu   |
| 3          | 3         | 17.56 | 3         | Vasilescu |
| 4          | 4         | 12.34 | 4         | Georgescu |
| NULL       | NULL      | NULL  | 2         | Popescu   |

# Subquery

- O “subinterogare” este o interogare de tip SELECT utilizata ca operand intr-o alta interogare
- O “subinterogare” poate fi privit ca un tabel temporar si tratat ca atare (inclusiv cu JOIN) eventual cu atribuire de nume (Alias) daca este nevoie
- Exemple
  - `SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);`

# Subquery

- Subquery – un instrument foarte puternic
- permite selectii in doua sau mai multe etape
  - o prima selectie **dupa un criteriu**
  - urmata de o doua selectie **dupa un alt criteriu** in **rezultatele primei selectii**
  - ... samd
- Exista restrictii asupra tabelelor implicate pentru evitarea prelucrarilor recursive (bucle potential infinite)
  - ex: UPDATE tabel1 SET ... SELECT ... FROM tabel1 nu este permis

# Subquery

- Subquery – un instrument foarte puternic
- Permite evitarea multor prelucrari PHP si trimiterea lor spre server-ul MySql
  - `INSERT INTO tabel1 ... SELECT ... FROM tabel2` permite inserarea printr-o singura interogare a mai multor linii in tabel1 (in functie de numarul de linii rezultate din tabel2)

# Contact

- Laboratorul de microunde si optoelectronica
- <http://rf-opto.etti.tuiasi.ro>
- [rdamian@etti.tuiasi.ro](mailto:rdamian@etti.tuiasi.ro)