PLANUL LECTIEI 12 (2 ORE),

JOI 23.11.2021

AICI REAMINTIM DESPRE INCEPUTUL PROCESULUI CU DESIGNUL APLICATIEI

§1. INTEROGARI IMBRICATE / ВЛОЖЕННЫЕ ПОДЗАПРОСЫ INTEROGARI ASOCIERI - JOIN / ОБЪЕДИНЕНИЯ (JOIN)

§2. MySQL InnoDB vs MyISAM - motoare de stocare

§3. CHARACTER SET COLLATION

INTEROGARI IMBRICATE SIMPLE / ПРОСТЫЕ ВЛОЖЕННЫЕ ПОДЗАПРОСЫ

Interogarile imbricate Simple sunt folosite pentru a reprezenta un set de valori, care ar trebui examinate într-un **predicat IN**, care este ilustrat în următorul exemplu:

+ Options

id	name	dept	SALARY
100	Andrei	IT	6500.00
200	Anton	IT	6500.00
300	Maxim	Marketing	6500.00
400	Dimon	Marketing	5500.00
500	Anton	Marketing	5000.00
501	Anita	Marketing	5000.00

Select id From bbb Where dept='Marketing' →

+ Options

id
300
400
500
501

Iar interogarea ce urmează,

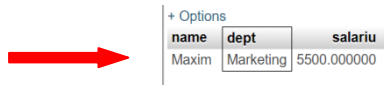
Select id, name, dept, salary
From bbb
Where id IN (Select id From bbb Where dept='Marketing') →

+ Options

id	name	dept	salary
300	Maxim	Marketing	6500.00
400	Dimon	Marketing	5500.00
500	Anton	Marketing	5000.00
501	Anita	Marketing	5000.00

Este echivalentă cu interogarea
Select id, name, dept, salary
From bbb
Where id IN (300,400,500,501)
Deci, atunci când SGBD-ul MySql procesează o cerere compusă/imbricată, **sistemul execută în primul rând subinterogarea imbricată**. Această interogare returnează un set de valori numerice **id,** ce corespunde **dept='Marketing'**, care furnizează anume setul (**300,400,500,501**).
Iata si un alt exemplu de utilizare a unei interogari imbricate cu o **functie agregata avg(salary)**

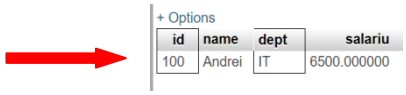
```
Select name, dept, avg(salary) AS  salariu
From bbb
Where id IN (Select id From bbb Where dept='Marketing')
Group by dept
```



name	dept	salariu
Maxim	Marketing	5500.000000

O subinterogare cu mai multe niveluri de imbricare poate fi ilustrată cu același exemplu.

```
Select id, name, dept, avg(salary) AS salariu
From bbb
        Where id IN
                (Select id From bbb
                        Where dept IN
                        (Select dept from bbb
                                Where Salary=6500))
group by dept
having(avg(salary))>5500
```



id	name	dept	salariu
100	Andrei	IT	6500.000000

iata si codul PHP, generat de PhpMyAdmin

```
$sql = "Select id, name, dept, avg(salary) AS  salariu\n"
    . "From bbb\n"
    . "Where id IN \n"
    . "                         (Select id From bbb \n"
    . "                         Where dept IN\n"
    . "                                 (Select dept from bbb where Salary=6500))\n"
    . "group by dept\n"
    . "having(avg(salary))>5500";
```

Iata rezultatul fara having(avg(salary))>5500

id	name	dept	salariu
100	Andrei	IT	6500.000000
300	Maxim	Marketing	5500.000000

Să urmarim pe pasi. Initial relatia bbb are forma:

id	name	dept	SALARY
100	Andrei	IT	6500.00
200	Anton	IT	6500.00
300	Maxim	Marketing	6500.00
400	Dimon	Marketing	5500.00
500	Anton	Marketing	5000.00
501	Anita	Marketing	5000.00

Pasul 1

```
Select id, dept from bbb
                        Where Salary=6500
```

id	dept
100	IT
200	IT
300	Marketing

Pasul 2

```
Select id From bbb
                Where dept IN ('IT','IT','Marketing')
```

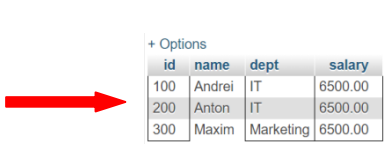
id
100
200
300
400
500
501

```
Select id, name, dept, avg(salary) AS salariu
From bbb
        Where id IN (100,200,300,400,500,501)
group by dept
having(avg(salary))>5500
```

id	name	dept	salariu
100	Andrei	IT	6500.000000

Ultimul SELECT, cel mai exterior, calculează rezultatul final de mai sus. În general, este permisă orice adâncime de imbricare a subinterogării.

```
Select id, name, dept,salary
From bbb
        Where id IN
                (Select id From bbb
                        Where salary IN
                        (Select salary from bbb
                                Where dept='IT      '))
```



+ Options

id	name	dept	salary
100	Andrei	IT	6500.00
200	Anton	IT	6500.00
300	Maxim	Marketing	6500.00

Același rezultat poate fi obținut folosind "big Mac"-ul /e vorba de acelasi tabel/

```
Select DISTINCT a.id, a.name, a.dept, a.salary
From bbb a, bbb b, bbb c
Where a.id=b.id  and  b.salary=c.salary  and  c.dept='IT      '
```

Cu o simplă mențiune, atunci când utilizăm unul și același tabel, să utilizăm Aliasurile si opțiunea DISTINCT, precum și faptul că aliasurile sunt definite numai într-o singură cerere.

La executarea acestei interogări compacte, sistemul trebuie să proceseze simultan date din trei tabele, în timp ce în exemplul anterior, aceste tabele sunt procesate la rândul lor. Bineînțeles, implementarea lor necesită resurse de memorie și timp diferite, dar acest lucru nu poate fi resimțit atunci când se lucrează cu o cantitate limitată de date în baza de date.

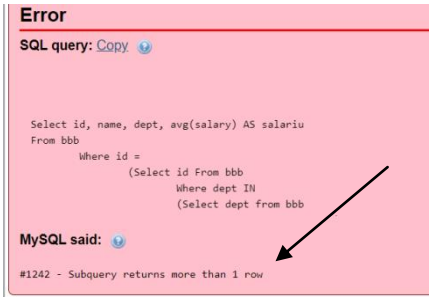
```
Select a.id, a.name, a.dept, a.salary
From bbb a
        Where a.id IN
                (Select b.id From bbb b
                        Where b.salary IN
                        (Select c.salary from bbb c
                                Where c.dept='IT      '))
```

+ Options

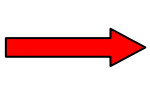
id	name	dept	salary
100	Andrei	IT	6500.00
200	Anton	IT	6500.00
300	Maxim	Marketing	6500.00

SUBINTEROGARE IMBRICATĂ CU UN OPERATOR DE COMPARAȚIE ALTUL DECÂT IN

Alți operatori de comparație (<>, <=, <,> = sau>) pot fi utilizați în astfel de interogări, cu toate acestea, dacă subinterogarea imbricată **returnează mai multe valori și operatorul IN** nu este utilizat, va apărea o eroare.



Select id, dept from bbb
Where Salary=6500 and name='Anton'



+ Options

id	dept
200	IT

Select id, name, Salary From bbb
Where dept = ('IT')

+ Options

id	name	Salary
100	Andrei	6500.00
200	Anton	6500.00

Describe aaa;
Describe bbb;

```
describe aaa
```

+ Options

Field	Type	Null	Key	Default	Extra
name	varchar(16)	YES		NULL	
aaa	int(11)	YES		NULL	
bbb	int(11)	YES		NULL	

Your SQL query has been executed successfully.

```
describe bbb
```

+ Options

Field	Type	Null	Key	Default	Extra
id	int(11)	YES		NULL	
name	varchar(16)	YES		NULL	
dept	varchar(16)	YES		NULL	
SALARY	double(6,2)	YES		NULL	

ALTER TABLE bbb
ADD PRIMARY KEY (ID);

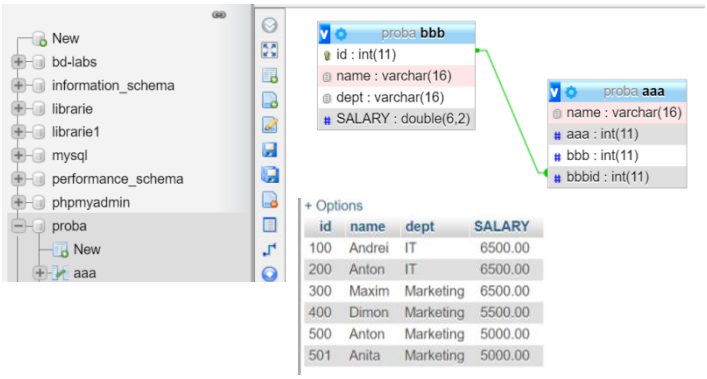
Your SQL query has been executed successfully.

```
DESCRIBE BBB
```

+ Options

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	
name	varchar(16)	YES		NULL	
dept	varchar(16)	YES		NULL	
SALARY	double(6,2)	YES		NULL	

ALTER TABLE aaa
ADD COLUMN bbbid INT;
ALTER TABLE aaa
ADD FOREIGN KEY (bbbid) REFERENCES bbb(id);



+ Options

id	name	dept	SALARY
100	Andrei	IT	6500.00
200	Anton	IT	6500.00
300	Maxim	Marketing	6500.00
400	Dimon	Marketing	5500.00
500	Anton	Marketing	5000.00
501	Anita	Marketing	5000.00

+ Options

name	aaa	bbb	bbbid
John	20	5	NULL
Ana	22	8	NULL
Wanda	32	8	NULL
Ion	17	3	NULL

<u>Modificam datele relatiei/tabelei</u> aaa

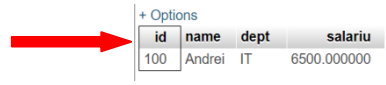
UPDATE aaa
SET bbbid = 500,
WHERE bbb=8;

+ Options

name	aaa	bbb	bbbid
John	20	5	100
Ana	22	8	500
Wanda	32	8	500
Ion	17	3	200

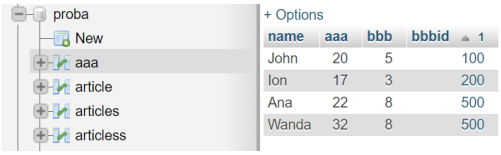
SUBINTEROGARI IMBRICATE CORELATE

```
Select id, name, dept, avg(salary) AS salariu
From bbb
        Where id IN (100,200,300,400,500,501)
group by dept
having(avg(salary))>5500
```



+ Options

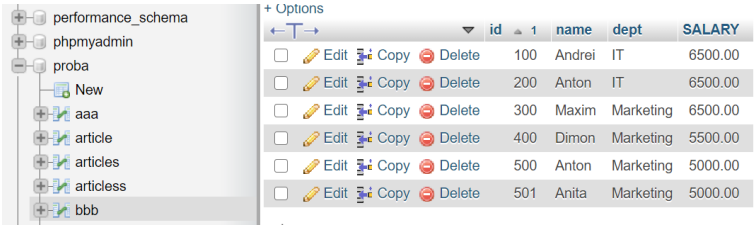
id	name	dept	salariu
100	Andrei	IT	6500.000000



proba
- New
- aaa
- article
- articles
- articless

+ Options

name	aaa	bbb	bbbid
John	20	5	100
Ion	17	3	200
Ana	22	8	500
Wanda	32	8	500

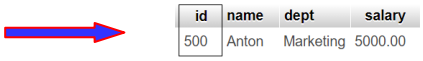


performance_schema
phpmyadmin
proba
- New
- aaa
- article
- articles
- articless
- bbb

+ Options

	id	name	dept	SALARY
Edit Copy Delete	100	Andrei	IT	6500.00
Edit Copy Delete	200	Anton	IT	6500.00
Edit Copy Delete	300	Maxim	Marketing	6500.00
Edit Copy Delete	400	Dimon	Marketing	5500.00
Edit Copy Delete	500	Anton	Marketing	5000.00
Edit Copy Delete	501	Anita	Marketing	5000.00

```
Select a.id, a.name, a.dept, a.salary
From bbb a
Where a.id IN
        (Select b.bbbid
                From aaa b Where b.bbbid=500)
```



id	name	dept	salary
500	Anton	Marketing	5000.00

Pasul 1.

```
        Select b.bbbid
                From aaa b Where b.bbbid=500
```

aaa
article
articles
articless

+ Options

bbbid
500
500

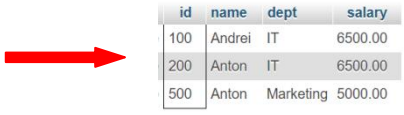
Pasul 2.

```
Select a.id, a.name, a.dept, a.salary
From bbb a
        Where a.id IN (500, 500)
```

id	name	dept	salary
500	Anton	Marketing	5000.00

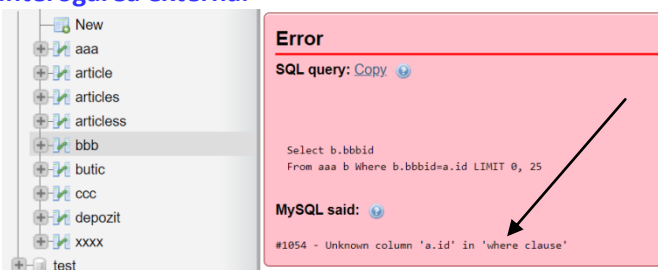
Sau altă interogare după cum urmează

```
Select a.id, a.name, a.dept, a.salary
From bbb a
Where a.id IN
        (Select b.bbbid
                From aaa b Where b.bbbid=a.id)
```



id	name	dept	salary
100	Andrei	IT	6500.00
200	Anton	IT	6500.00
500	Anton	Marketing	5000.00

Atentie! În acest caz, interogarea imbricată nu lucrează, fiindcă ea depinde de elementul a.id definit în interogarea externă.



New
- aaa
- article
- articles
- articless
- bbb
- butic
- ccc
- depozit
- xxxx
- test

Error

SQL query: Copy

```
Select b.bbbid
From aaa b Where b.bbbid=a.id LIMIT 0, 25
```

MySQL said:

```
#1054 - Unknown column 'a.id' in 'where clause'
```

Această interogare compusă diferă de cea discutată mai sus prin aceea că subinterogarea imbricată nu poate fi procesată înainte ca subinterogarea externă să fie procesată. Acest lucru se datorează faptului că subinterogarea imbricată depinde de valoarea **a.id**, care se modifică pe măsură ce sistemul verifică diferite rânduri în tabelul **bbb**. Prin urmare, din punct de vedere conceptual, procesarea se realizează după cum urmează:

1. Sistemul verifică primul rând al tabelului **bbb**. Să presupunem că aceasta este o linie cu numărul **id=100.** Atunci valoarea **b.bbbid** va avea si ea valoarea de **100**. In acest caz sistemul va procesa interogarea internă

   Select b.bbbid
   From aaa b Where b.bbbid=100

   rezultând mulțimea (100). Acum sistemul poate finaliza procesarea pentru furnizorul numărul 100. Selectarea valorilor de **a.id, a.name, a.dept, a.salary** va fi efectuată dacă și numai dacă **a.id = 100** aparține acestui set, ceea ce este evident adevărat.

2. Apoi, sistemul va repeta acest tip de procesare pentru următorul **a.id** etc. până când toate rândurile din tabelul **bbb** au fost revizuite.

Aceste ***subinterogări imbricate*** sunt denumite ***subinterogări imbricate corelate*** deoarece rezultatul lor depinde de valorile definite în interogarea externă. Prin urmare, procesarea corelată a subinterogării trebuie repetată pentru fiecare valoare extrasă din subinterogarea externă, mai degrabă decât efectuată o dată pentru totdeauna.

Să vedem un exemplu de utilizare a aceluiași tabel bbb într-o subinterogare exterioară și o subinterogare imbricată corelată.
(relatia initiala bbb

bbb a **bbb b**

```
Select DISTINCT a.id, a.name, a.dept, a.salary
From bbb a
Where a.name not IN
      (Select b.name
            From bbb b
                  Where b.salary<>a.salary)
```

id	name	dept	salary
100	Andrei	IT	6500.00
300	Maxim	Marketing	6500.00
400	Dimon	Marketing	5500.00
501	Anita	Marketing	5000.00

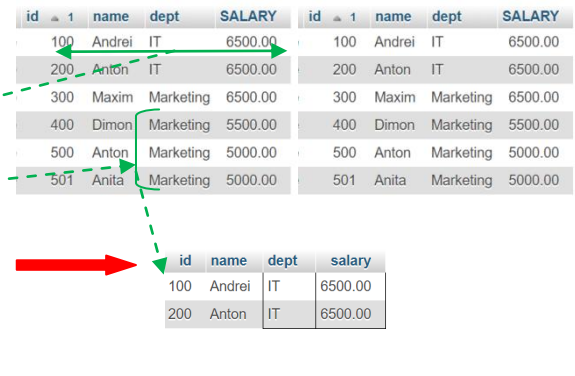
Id=100, name=Andrei, Salary=6500 atunci interogarea imbricată (internă) va fi
Select b.name From bbb b Where b.salary<>6500 rezultatul interogarii

name
Dimon
Anton
Anita

Id=200, name=Anton, Salary=6500 atunci interogarea imbricată (internă) va fi
Select b.name From bbb b Where b.salary<>6500 rezultatul interogarii

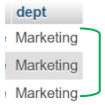
....

Sau o altă variantă:




```
Select DISTINCT a.id, a.name, a.dept, a.salary
From bbb a
Where a.dept not IN
        (Select b.dept
                From bbb b
                        Where b.salary<>a.salary)
```

Id=100, dept='IT ', Salary=6500 atunci interogarea imbricată (internă) va fi
Select b.dept From bbb b Where b.salary<>6500 rezultatul interogarii
Id=200, dept='IT ', Salary=6500



....

INTEROGĂRI CU EXISTS (există)

Reamintim

LOGICA AFIRMAŢIILOR CUANTIFICATE
Simbolul ∧ înseamnă „pentru toate" şi se numeşte cuantificatorul universal,
AU: „∀ x din D, Q(x)".
Simbolul ∃ înseamnă „există" şi se numeşte cuantificatorul existenţei.
AE : „∃x în D, astfel încât P(x)".

DESCRIERE

Subinterogările care utilizează cuvântul cheie **EXISTS** vor returna valoarea ***True***, dacă subinterogarea returnează măcar un rând din tabel. În schimb, subinterogările care folosesc **NOT EXISTS** vor returna ***True*** numai dacă interogarea nu returnează niciun rând din tabel.

***Subinterogările* EXISTS** *ignoră coloanele specificate de SELECTul subinterogării*, deoarece nu sunt relevante. (Pe "El" îl interesează, *subinterogarea* imbricată intoarce sau nu rinduri/inscrieri?!) De exemplu,

SELECT col1 FROM tabl1 WHERE EXISTS (SELECT * FROM tabl2);

și

SELECT col1 FROM tabl1 WHERE EXISTS (SELECT col2 FROM tabl2);
produc rezultate identice.

Cuantificatorul EXISTS (există) este un concept împrumutat din logica formală. În SQL, un predicat cu un cuantificator existențial este reprezentat de expresia EXISTS si are froma:

SELECT col1 FROM tabl1 WHERE EXISTS (SELECT * FROM tabl2);
Sau
SELECT col1 FROM tabl1 WHERE EXISTS (SELECT col2 FROM tabl2);

O astfel de expresie este considerată adevărată numai atunci când rezultatul evaluării „SELECT * FROM ...” este un set ne-gol, adică când există măcar o înregistrare în tabelul specificat în clauza FROM a subinterogării care satisface clauza WHERE. (În practică, această interogare va fi întotdeauna un set corelat!!.)

```
Select DISTINCT a.id, a.name, a.dept, avg(a.salary) As SAL_MEDIU
From bbb a
Where EXISTS
        (Select b.dept
                From bbb b
                        Where b.name<>'Anton')
GROUP BY A.dept
HAVING avg(a.salary)>5000
ORDER BY name
```

id	name	dept	SAL_MEDIU
100	Andrei	IT	6500.000000
300	Maxim	Marketing	5500.000000

In particular rezultatul interogarii imbricate
Select b.dept From bbb b Where b.name<>'Anton' este nevid

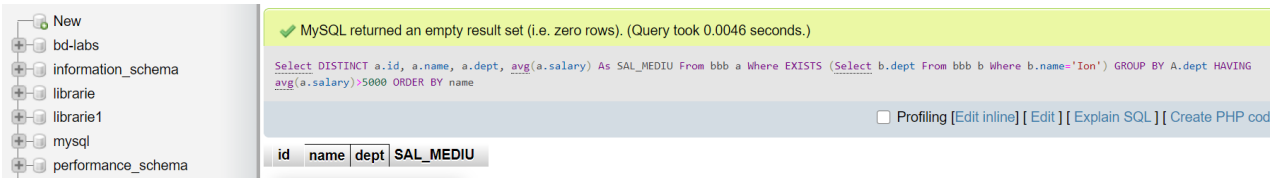
dept
IT
Marketing
Marketing
Marketing

STUDIUL DE CAZ 2. INTEROGAREA IMBRICATĂ NU ARE INSCRIERI

```
        Select b.dept
                From bbb b
                        Where b.name='Ion')
```

Atunci si relatia externă nu are nimic

```
Select DISTINCT a.id, a.name, a.dept, avg(a.salary) As SAL_MEDIU
From bbb a
Where EXISTS
        (Select b.dept
                From bbb b
                        Where b.name='Ion')    /poate fi si a.nume='Ion'/
GROUP BY A.dept
HAVING avg(a.salary)>5000
ORDER BY name
```



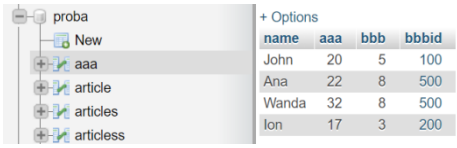
Sistemul selectează secvențial rândurile din tabelul **bbb a**, extrage valorile coloanelor **a.id, a.name, a.dept** din acesta și apoi verifică dacă condiția de existență **EXISTS** este adevărată, adică tabelul **bbb b** conține cel puțin un rând cu valoarea **b.name='Ion'** selectată din tabelul **bbb b (sau bbb a)**? Dacă condiția este îndeplinită, valoarea rezultată a coloanelor **a.id, a.name, a.dept** sunt incluse în rezultat.

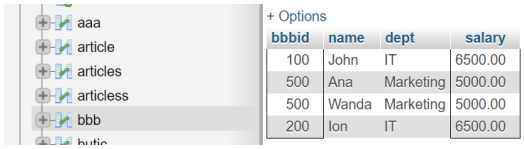
Deși acest prim exemplu se arată doar un mod diferit de formulare a unei interogări pentru o problemă care poate fi rezolvată în alte moduri (folosind operatorul **IN** sau **JOIN**), **EXISTS** este una dintre cele mai importante caracteristici ale **SQL**. De fapt, orice interogare care este exprimată în termeni de **IN** poate fi formulată alternativ și utilizând **EXISTS**. Cu toate acestea, opusul nu este adevărat.

FUNCȚII ÎNTR-O SUBINTEROGARE

Acum, după ce ne-am familiarizat cu diferitele formulări de subinterogări și aliasuri imbricate, este mai ușor să înțelegem textul și algoritmul de implementare a cererii CU FUNCTII AGREGATE in interogarile imbricate

```
Select b.bbbid, b.name, a.dept, a.salary
From aaa b, bbb a
      Where b.bbbid=a.id  and a.salary>=
                  (Select avg(a.salary)
                  from bbb a
                  Where a.id=b.bbbid)
```






```
SELECT b.bbbid, b.name, a.dept, a.salary
From aaa b, bbb a
Where b.bbbid=a.id and a.salary>=
                  (SELECT avg(a.salary)
                  from bbb a
                  group by a.dept
                  having(avg(a.salary))>=6500)
```

Bineînțeles, aceasta este o subinterogare corelată: aici, în primul rând, se determină valoarea medie a salariului care face parte din **aaa b** și abia apoi se află celelalte detalii.

Cu acest exemplu, voi încheia informarea cu privire la interogări imbricate oferind o serie de exemple de interogări folosind mecanismul acestora.

id_autor	nume_autor
1	Petru
2	Diuma
3	Vieru
4	Lena
5	Ioana
6	Ghita
7	Colea
8	Nicu
9	Valea

idcarte	autor	titlu	pret	cantitatea	id_autor
2-2222-222-10	Petru GFGF	Laborator Mysql-Php vbvbvb	950.00	23	1
2-2222-222-13	Vieru	MAMA	2000.00	200	3
2-2222-2222-1	Diuma	Cei trei muschetari Php	775.00	10	1
2-2222-2222-2	Diuma	Cei trei muschetari Php	775.00	10	1
2-2222-2222-4	Petru	Cei trei muschetari Php	155.00	20	1
2-2222-2222-5	Ion	Cei trei muschetari Php	310.00	5	3
2-2222-2222-6	Vica	Cei trei muschetari Php	930.00	40	4
2-2222-2222-7	Sandu	Cei trei muschetari Php	465.00	222	2
2-2222-2222-8	Eminescu	Ghid Php	1000.00	3	6

Autor Carti

```
Select autor, sum(cantitatea) As Numar_Carti
From Carti
Group by Autor

Select autor
From (Select autor, sum(cantitatea) As Numar_Carti
        From Carti Group by Autor)
As Rezultat
Where numar_carti>=20

Select *
From carti A
Where A.cantitatea> (Select avg(B.cantitatea) From carti B)
From (Select autor, sum(cantitatea) As Numar_Carti
        From Carti Group by Autor)
```

Sarcină:
Este cunoscut numele autorului “Petru” care are scrise mai multecarti, dar nu cunoastem Id_autor lui. Este necesar de obtia obtinut informatia despre acest autor (informatia deplina)

```
Select *
From autor
Where id_autor=(Select id_autor From carti where
autor='Petru')

Select *
From autor
Where id_autor=(Select DISTINCT id_autor From carti where
autor='Petru')

Select *
From autor
Where id_autor IN (Select DISTINCT id_autor From carti where
autor='Petru')
```

```
Select *
From autor A
Where not Exists (Select * from autor B
      Where b.nume_autor='Petru')

Select *
From autor A
Where A.nume_autor='Petru' and EXISTS (Select * from carti B
      Where b.autor=a.nume_autor)
```

(in locul =, Any (>=Any, <=Any, =Any), All, Some)

Sa determinam N-rul de carti a autorilor din BD "librarie"

Sa determinam N-rul de carti a autorilui 'XXXX' din BD "librarie"

Sa determinam lista autorilor care au scrise mai multe de M carti

```
Select autor
From (Select autor, sum(cantitatea) as Numar_Carti From Carti
Group by autor) As rezultate
Where Numar_c>M
```

De prezentat informatia despre toate cartile din biblioteca numarul carora (cantitatea) este mai mare decit AVG valaorea media a numarului de carti la momentul dat.

```
Select * from carti A
Where A.cantitatea>(Select avg(cantitatea) from carti b)
```

HAVING – filtrarea pe grupuri
WHERE – filtreaza linii/inscrierele in relatii

Exemple

1. **Cartile**
2. **Produsele**
3. **Examenele**

Cum sa evitam erorile cind utilizam interogarile imbricate

INTEROGARI ASOCIERI - JOIN

https://ro.qaz.wiki/wiki/Join_(SQL)

Ce este o asociere?

O asociere este o conexiune **între două tabele R** și **S** în care cele două tabele sunt ***îmbinate în funcție de un câmp*** (***atribut comun, atribut de asociere/compunere***) pe care îl au în comun, creând un nou tabel **T** virtual (care poate fi salvat ca tabel real), cu o structură complex format implicit din mulțimea atributelor tabelelor sursă.

O clauză SQL **JOIN** - corespunzătoare unei operații de **JOIN** în algebră relațională - **combină coloane dintr-una sau mai multe tabele într-o bază de date relațională** . Se creează un set care poate fi salvat ca tabel sau folosit așa cum este.

JOIN este un mijloc de combinare a coloanelor dintr-unul (auto-îmbinare) sau mai multe tabele utilizând valori comune fiecăruia. Standardul SQL specifică cinci tipuri de JOIN:

1. **INNER,**
2. **LEFT JOIN,**
3. **RIGHT JOIN,**
4. **FULL JOIN,**
5. **CROSS JOIN.**
6. **LEFT [OUTER[JOIN,**
7. **RIGHT [OUTER] JOIN,**
8. **FULL [OUTER[JOIN.**

Ca un caz special, un tabel (tabel de bază, vedere sau tabel îmbinat) poate fi JOIN însuși într-o auto-Unire.

1. INNER JOIN
2. RIGHT JOIN
3. LEFT JOIN
4. FULL JOIN
5. LEFT OUTER JOIN с исключением записей
6. RIGHT OUTER JOIN с выбором уникальных
7. FULL OUTER JOIN с выбором уникальных записей
8. CROSS JOIN
9. NATURAL JOIN

Pentru realizarea Asocierilor, este necesar:

1. **Minimum 2 tabele**
2. Cimpurile dupa care urmează să fie efectuată asocierea
3. utilizarea comenzii **SELECT** cu una din clauzele **JOIN** de mai sus,
4. **urmată** de o conditie de asociere (uneori mai multe) si care este expusa in conditia prezentata de propozitia **ON <conditie>,**
5. adica daca conditia este **TRUE**, are loc asocierea, daca este FALSE, nu are loc asocierea.

SINTAXA GENERALĂ:

```
            SELECT <lista>
            FROM <Nume tabela 1>
[cuvint cheie] JOIN  <Nume tabela 2>
            ON <Criteriu de asociere> USING <clauze>
            [criteriu de selectie WHERE <conditie>]
            [ORDER By <cimpuri>]
```

Unde

[cuvint cheie] – ***OUTER, NATURAL, RIGHT, FULL, CROSS, LEFT***

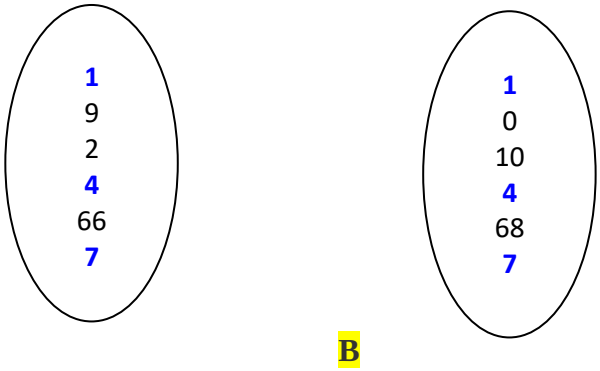
Există 2 abordari in explicarea cum lucrează JOIN.

a. **Folosind teoria multimilor (operatiile R∩ $S - union, R \cup S - intersect, R/S - except$)**
b. **Operatii cu diagrame**

Important pentru operatiile JOIN:

1. **Operatia JOIN nu se utilizeza pe un singur tabel/relatie**
2. **Operatia JOIN urmează să fie permisă de Administratorul BD, fiindcă utilizînd JOIN am putea afecta securitatea datelor /furt de date/**
3. **Administratorul BD poate crea pentru user operatiile JOIN necesare si astfel a păstra securitatea datelor BD**

Idea de asociere ia inceputul in teoria multimilor.



A **B**

Asocierea A cu B inseamna o noua relatie C, care contine elementele **din A si B commune**. Or, Prin Asocierea a 2 relatii, **noi nu vom asocia inscrierele lor, ci doar numai anumite coloane ce satisfac unor conditii de asociere expuse in clause de tip ON <conditie>.**

- **Este evident ca aceste conditii pot fi impuse atât atributelor cheie cât și celor non-cheie.**
- **Este evident ca rezultatul Asocierilor va depinde, atât de tipul Asocierii, cât și de conditia din clauza ON.**

Deoarece am vorbit despre asocieri pornind de la teoria relatiilor definite prin teoria multimilor, pentru o intelegere mai exemplificată vom folosi diagramele lui **Venn/Eiler. Dar Atentie, acest lucru poate fi acceptat doar pentru perceperea operatiei JOIN, genezei ei (***geneză sf [At: ODOBESCU, S. III, 624 / Pl: ~ze / E: fr genèse] 1 Proces de naștere și de formare a unei ființe sau a unui lucru Si: naștere, origine. 2 Ansamblu de fapte, de împrejurări, de elemente care au concurat la formarea unui lucru.). Doar atît, fiindcă în realitate lucrurile sunt mult mai complexe.*

Sa urmărim tipurile de Join de mai sus, utilizînd Diagramele lui Venn. Atentionam că astfel de cazuri cum sunt precautate cele de mai sus, pentru cazul relatiilor A si B, **par a fi mai des ipotetice,** cind relatiile au acelasi **grad** */Numărul de atribute ale unui tabel sau relații reprezintă* ***gradul relației (PUTEREA RELATIEI – степень отношения,*** **cardinalitatea** */МОЩНОСТЬ ОТНОШЕНИЯ – numărul de tupluri/inscrieri/ЗАПИСЕЙ dintr-o* **relație/** si **Schemele sunt compatibile**, dacă ele au acelasi grad si atributele au acelasi tip.

Deci in concluzie, exemplele de relatii pe care le vom precauta pentru JOIN, sunt "relatiile cu schemele compatibile" ce se intilnesc in practica mai rar, dar in continuare vom precauta si alte cazuri.

Cazul 1. (INNER JOIN)

Vom precauta pentru asociere 2 tabele A si B nelegate intre ele

Tabel_A

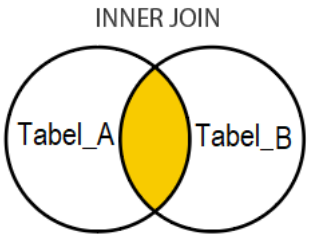
Id	Name
1	Ion
2	Nata
3	Petru
4	Ghita

```
CREATE TABLE Tabel_A (
id int not null,
name varchar(20)
);
```

Tabel_B

Id	Name
1	Nicolai
2	Ion
3	Elena
4	Petru

```
SELECT *
FROM TABEL_A
INNER JOIN TABEL_B
        ON TABEL_A.NAME=TABEL_B.NAME
```



Rezultatul

Id	Name	Id	Name
1	Ion	2	Ion
3	Petru	4	Petru

INNER JOIN – prezintă înscrierile care sunt comune atât **TABEL_A** si **TABEL_B**, porneste de la tabela prezenta in **FROM** si asociază la ea ceea ce urmează în **INNER JOIN**.

Cazul 2. (FULL OUTER JOIN)

Vom precauta pentru asociere 2 tabele A si B nelegate intre ele

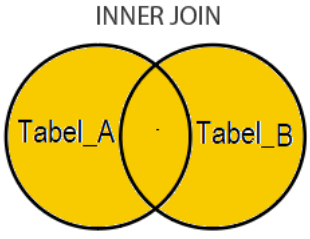
Tabel_A

Id	Name
1	Ion
2	Nata
3	Petru
4	Ghita

Tabel_B

Id	Name
1	Nicolai
2	Ion
3	Elena
4	Petru

```
SELECT *
FROM TABEL_A
FULL  OUTER  JOIN TABEL_B
        ON TABEL_A.NAME=TABEL_B.NAME
```



Rezultatul

Id	Name	Id	Name
1	Ion	2	Ion
2	Nata	Null	Null
3	Petru	4	Petru
4	Ghita	Null	Null
Null	Null	1	Nicolai
Null	Null	3	Elena

FULL OUTER JOIN - asociază toată înscrierile din **TABEL_A** și **TABEL_B** cu posibilitatea de coincidență a tabelei din stinga, **TABEL_A**, cu cei din dreapta **TABEL_B**. Dacă coincidențe nu sunt, în partea vidă apare **NULL**.

Cazul 3. (LEFT OUTER JOIN)

Vom precauta pentru asociere 2 tabele A si B nelegate intre ele

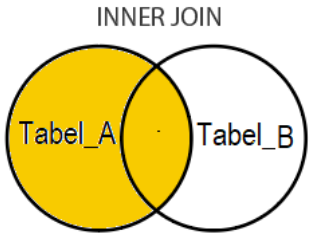
Tabel_A

Id	Name
1	Ion
2	Nata
3	Petru
4	Ghita

Tabel_B

Id	Name
1	Nicolai
2	Ion
3	Elena
4	Petru

```
SELECT *
FROM TABEL_A
LEFT  OUTER  JOIN TABEL_B
        ON TABEL_A.NAME=TABEL_B.NAME
```



Rezultatul

Id	Name	Id	Name
1	Ion	2	Ion
2	Nata	Null	Null
3	Petru	4	Petru
4	Ghita	Null	Null

LEFT OUTER JOIN - asociază toată înscrierile din **TABEL_A** și **TABEL_B** cu posibilitatea de coincidență a tabelei din stinga, **TABEL_A**, cu cei din dreapta **TABEL_B**. Dacă coincidențe nu sunt, în rezultatul final apare **NULL**.

Cazul 4. (RIGHT OUTER JOIN)

Vom precauta pentru asociere 2 tabele A si B nelegate intre ele

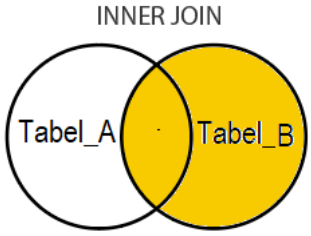
Tabel_A

Id	Name
1	Ion
2	Nata
3	Petru
4	Ghita

Tabel_B

Id	Name
1	Nicolai
2	Ion
3	Elena
4	Petru

```
SELECT *
FROM TABEL_A
RIGHT  OUTER  JOIN TABEL_B
        ON TABEL_A.NAME=TABEL_B.NAME
```



Rezultatul

Id	Name	Id	Name
Null	Null	1	Nicolai
4	Ion	2	Ion
Null	Null	3	Elena
3	Petru	4	Petru

RIGHT OUTER JOIN - asociază toată înscrierile din **TABEL_B** și **TABEL_A** cu posibilitatea de coincidență a tabelei din stinga, **TABEL_B**, cu cei din dreapta **TABEL_A**. Dacă coincidențe nu sunt, în rezultatul final apare **NULL**.

EXERCITIU:
Să se prezinte acele cîmpuri ce nu se regasesc in ambele tabele TABEL_A și TABEL_B

SOLUTIE

Vom precauta pentru asociere 2 tabele A si B nelegate intre ele

Tabel_A

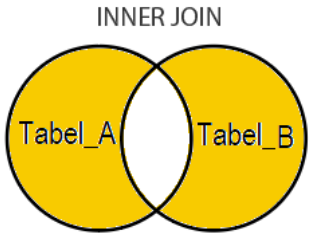
Id	Name
1	Ion
2	Nata
3	Petru
4	Ghita

Tabel_B

Id	Name
1	Nicolai
2	Ion
3	Elena
4	Petru

```
SELECT *
FROM TABEL_A
FULL  OUTER  JOIN TABEL_B

      ON TABEL_A.NAME=TABEL_B.NAME
      WHERE TABEL_A.ID IS NULL OR TABEL_B.ID IS NULL
```



Rezultatul

Id	Name	Id	Name
2	Nata	Null	Null
4	Ghita	Null	Null

EXERCITIU:
Să se prezinte acele cîmpuri din TABEL_A care nu sunt in TABEL_B

EXERCITIU:
Să se prezinte acele cîmpuri din TABEL_B care nu sunt in TABEL_A

IN EXEMPLELE DE MAI SUS, Tabelele precautate au fost fără legatura între ele, doar pentru a demonstra posibilitatile diferitor comenzi de Asociere JOIN

Evident ca BD relationale sunt legate intre ele, adică intre ele exista anumite relatii. Vom precăuta in acest caz citeva exemple.Fie date 2 tabele, tabelul Părinte **Departament** si tabelul Copil **Angajat.**

Departament

Id	Name
1	IT
2	Programare
3	Tehnic
4	Finanțe
5	RU

Angajat

Id	Name	D_id
1	Vlad	1
2	Anton	2
3	Alex	5
4	Boris	2
5	Iurie	4

```
CREATE TABLE Departament (
id int not null primary key AUTO_INCREMENT,
name varchar(20)
) Engine=innodb;
```

id	name
1	IT
2	Programare
3	Tehnic
4	Finante
5	RU

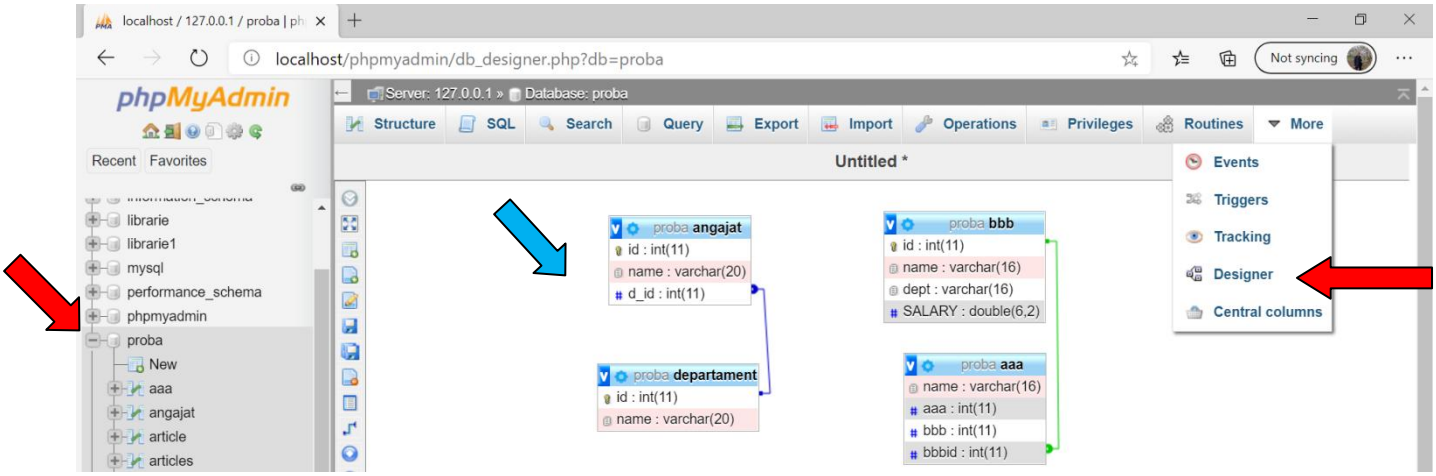
```
CREATE TABLE Angajat_1 (
id int not null primary key AUTO_INCREMENT,
name varchar(20),
d_id int not null,
    FOREIGN KEY (d_id)
    References Departament(id)
    ON UPDATE CASCADE
    ON DELETE CASCADE
) Engine=innodb;
```

id	name	d_id
1	Vlsd	1
2	Anton	2
3	Alex	5
4	Boris	2
5	Iurie	4

```
INSERT INTO `angajat_1`(`name`, `d_id`) VALUES
('Vlad',1),('Anton',2),('Alex',5),('Boris',2),('Iurie',4);
```

```
CREATE TABLE Angajat (
id int not null primary key AUTO_INCREMENT,
name varchar(20),
d_id int not null,
    FOREIGN KEY (d_id)
    References Departament(id)
) Engine=innodb;
```

```
INSERT INTO `angajat`(`name`, `d_id`) VALUES
('Vlsd',1),('Anton',2),('Alex',5),('Boris',2),('Iurie',4);
```



Exemplu 1.

```
Select A.id, A.name, B.name As Departament
From Angajat A
INNER JOIN Departament B
ON A.d_id=B.Id
```

id	name
1	IT
2	Programare
3	Tehnic
4	Finante
5	RU

Departament B

id	name	d_id
1	Vlsd	1
2	Anton	2
3	Alex	5
4	Boris	2
5	Iurie	4

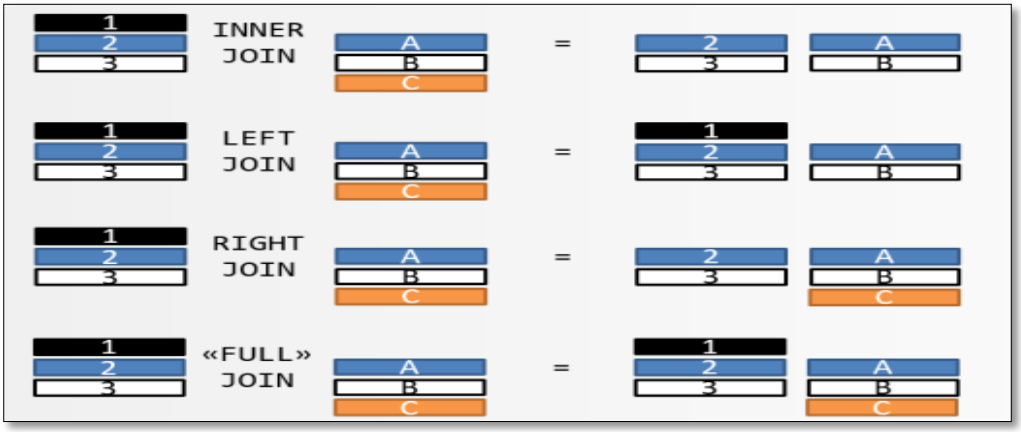
Angajat A

+ Options

id	name	Departament
1	Vlsd	IT
2	Anton	Programare
3	Alex	RU
4	Boris	Programare
5	Iurie	Finante

Rezultat

Pentru e lamuri acest caz vom folosi metoda digramelor!



Angajat

Id	Name	D_id
1	Vlad	1
2	Anton	2
3	Alex	5
4	Boris	2
5	Iurie	4

Departament

Id	Name
1	IT
2	Programare
3	Tehnic
4	Finanțe
5	RU
6	Securitate
7	Achizitii

Rezultat

Id	Name	Departament
1	Vlad	IT
2	Anton	Programare
3	Alex	RU
4	Boris	Programare
5	Iurie	Finanțe

Înserăm un rind nou in Departament după cum urmează

id	name
1	IT
2	Programare
3	Tehnic
4	Finante
5	RU
6	Securitate
7	Achizitii

Select A.id, A.name, B.name As Departament
From Angajat A
LEFT OUTER JOIN Departament B
ON A.d_id=B.Id

+ Options

id	name	Departament
1	Vlsd	IT
2	Anton	Programare
3	Alex	RU
4	Boris	Programare
5	Iurie	Finante

Angajat

Id	Name	D_id
1	Vlad	1
2	Anton	2
3	Alex	5
4	Boris	2
5	Iurie	4

Departament

Id	Name
1	IT
2	Programare
3	Tehnic
4	Finanțe
5	RU
6	Securitate
7	Achizitii

Rezultat

Id	Name	Departament
1	Vlad	IT
2	Anton	Programare
3	Alex	RU
4	Boris	Programare
5	Iurie	Finanțe

```
Select A.id, A.name, B.name As Departament
From Angajat A
RIGHT  OUTER  JOIN Departament B
ON A.d_id=B.Id
```

+ Options

id	name	Departament
1	Vlsd	IT
2	Anton	Programare
3	Alex	RU
4	Boris	Programare
5	Iurie	Finante
NULL	NULL	Tehnic
NULL	NULL	Securitate
NULL	NULL	Achizitii

Angajat

Id	Name	D_id
1	Vlad	1
2	Anton	2
3	Alex	5
4	Boris	2
5	Iurie	4

Departament

Id	Name
1	IT
2	Programare
3	Tehnic
4	Finanțe
5	RU
6	Securitate
7	Achizitii

Rezultat

Id	Name	Departament
1	Vlad	IT
2	Anton	Programare
3	Alex	RU
4	Boris	Programare
5	Iurie	Finanțe
Null	Null	Tehnic
Null	Null	Securitate
Null	Null	Achizitii

-- How to do
FULL [OUTER] JOIN in MySQL (MariaDB)

First method

```
SELECT * FROM t1
LEFT JOIN t2 ON t1.id = t2.id
UNION
SELECT * FROM t1
RIGHT JOIN t2 ON t1.id = t2.id
```

Second Method

```
-- The query above works for special cases where a FULL OUTER JOIN operation would not produce any
duplicate rows.
-- The query above depends on the UNION set operator to remove duplicate rows introduced by the query
pattern.
-- We can avoid introducing duplicate rows by using an anti-join pattern for the second query,
-- and then use a UNION ALL set operator to combine the two sets. In the more general case,
-- where a FULL OUTER JOIN would return duplicate rows, we can do this
  SELECT * FROM t1
  LEFT JOIN t2 ON t1.id = t2.id
  UNION ALL
  SELECT * FROM t1
  RIGHT JOIN t2 ON t1.id = t2.id
  WHERE t1.id IS NULL
```

```
Select *
From Angajat A
LEFT  OUTER  JOIN Departament B  ON A.d_id=B.Id
UNION all
Select *
From Angajat A
RIGHT  OUTER  JOIN Departament B ON A.d_id=B.Id
```

+ Options

id	name	d_id	id	name
1	Vlsd	1	1	IT
2	Anton	2	2	Programare
3	Alex	5	5	RU
4	Boris	2	2	Programare
5	Iurie	4	4	Finante
NULL	NULL	NULL	3	Tehnic
NULL	NULL	NULL	6	Securitate
NULL	NULL	NULL	7	Achizitii

Angajat

Id	Name	D_id
1	Vlad	1
2	Anton	2
3	Alex	5
4	Boris	2
5	Iurie	4

Departament

Id	Name
1	IT
2	Programare
3	Tehnic
4	Finanțe
5	RU
6	Securitate
7	Achizitii

Rezultat

Id	Name	D_id	id	Departament
1	Vlad	1	1	IT
2	Anton	2	2	Programare
3	Alex	5	5	RU
4	Boris	2	2	Programare
5	Iurie	4	4	Finanțe
Null	Null	Null	3	Tehnic
Null	Null	Null	6	Securitate
Null	Null	Null	7	Achizitii

Select A.id, A.name, B.name As Angajat
From Departament A
LEFT OUTER JOIN Angajat B
ON A.Id = B.d_id

+ Options

id	name	Angajat
1	IT	Vlsd
2	Programare	Anton
5	RU	Alex
2	Programare	Boris
4	Finante	Iurie
3	Tehnic	NULL
6	Securitate	NULL
7	Achizitii	NULL

Select A.id, A.name, B.name As Angajat
From Departament A
RIGHT OUTER JOIN Angajat B
ON A.Id = B.d_id

+ Options

id	name	Angajat
1	IT	Vlsd
2	Programare	Anton
5	RU	Alex
2	Programare	Boris
4	Finante	Iurie

Retineti!!

Note that UNION removes duplicates, FULL OUTER JOIN doesn't
Rețineți că UNION elimină duplicatele, FULL OUTER JOIN nu

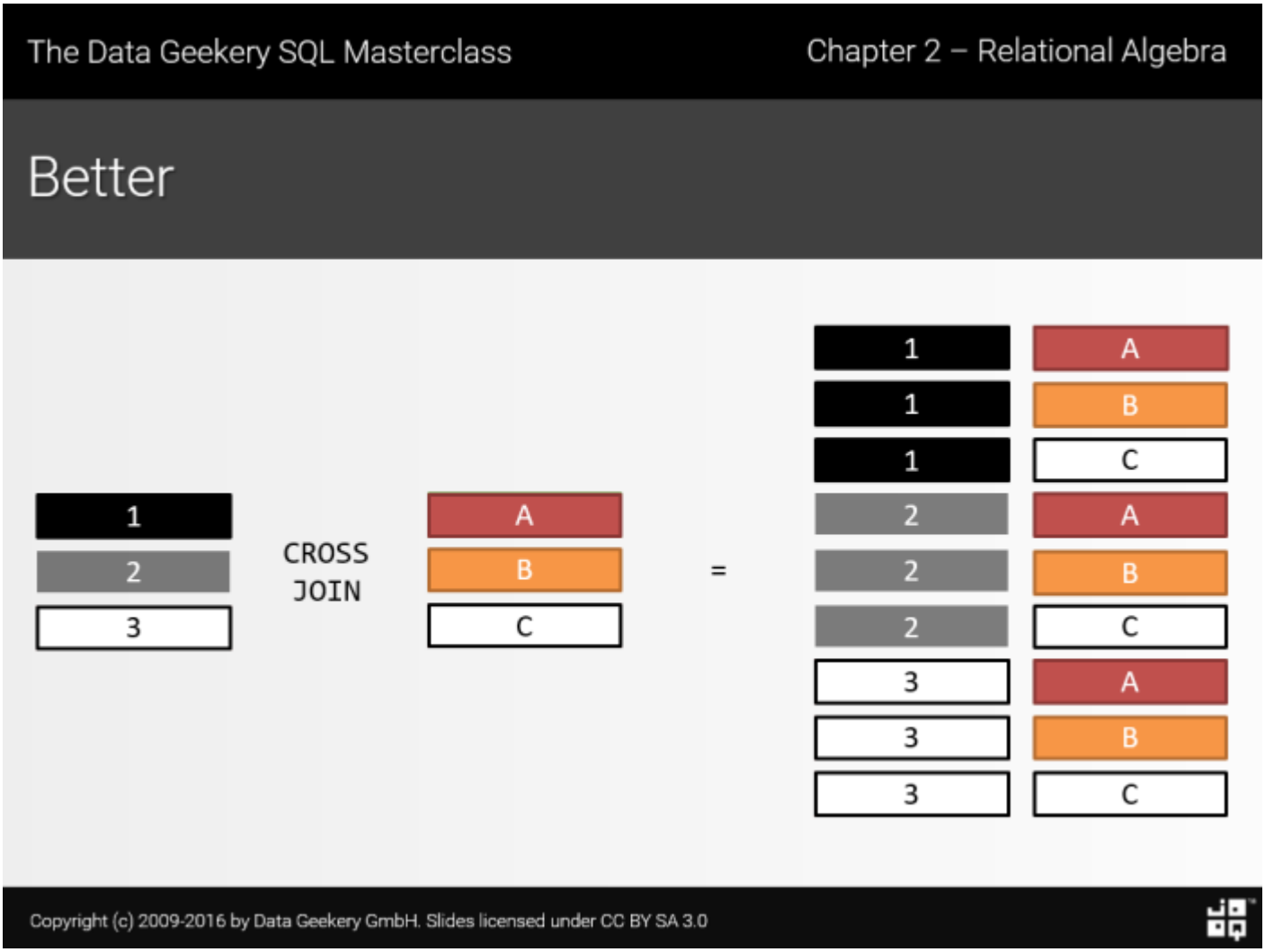
http://citforum.ru/database/sql_kg/index.shtml

```
Select A.id, A.name, B.name As Departament
From Angajat A
FULL  OUTER  JOIN Departament B
ON A.d_id=B.Id
```

So, what's a better way to illustrate JOIN operations?

https://blog.jooq.org/2016/07/05/say-no-to-venn-diagrams-when-explaining-joins/

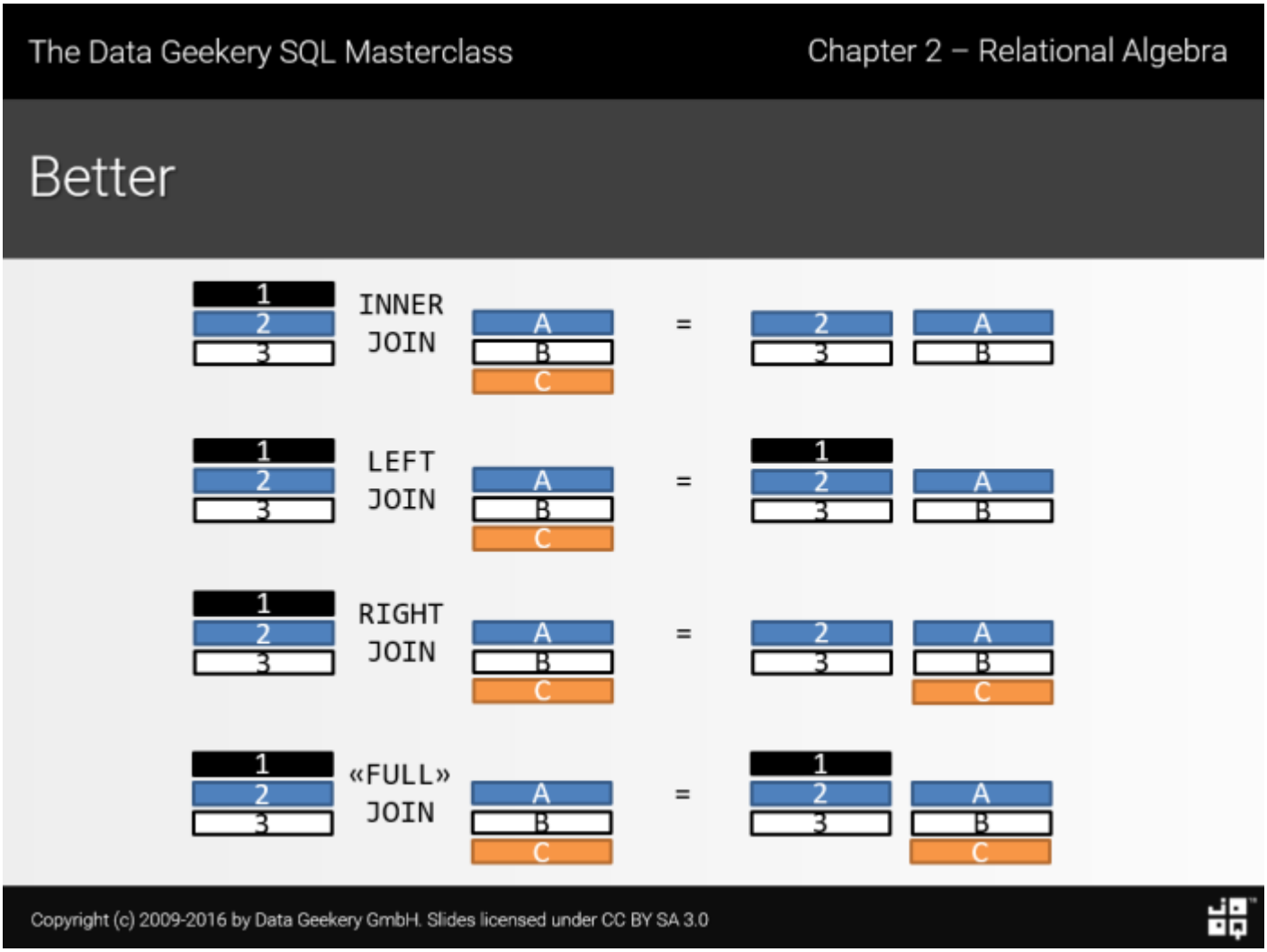
JOIN diagrams! Let's look at CROSS JOIN first, because all other JOIN types can be derived from CROSS JOIN:



Remember, in a cross join (in SQL also written with a comma separated table list, historically) is just taking every item on the left side, and combines it with every item on the right side. When you CROSS JOIN a table of 3 rows with a table of 4 rows, you will get 3×4=12 result rows. See, I'm using an "x" character to write the multiplication. I.e. a "cross".

INNER JOIN

All other joins are still based on cross joins, but with additional filters, and perhaps unions. Here's an explanation of each individual JOIN type.



In plain text, an INNER JOIN is a CROSS JOIN in which only those combinations are retained which fulfil a given predicate. For instance:

```
-- "Classic" ANSI JOIN syntax
SELECT *
FROM author a
JOIN book b ON a.author_id =
b.author_id


-- "Nice" ANSI JOIN syntax
SELECT *
FROM author a
JOIN book b USING (author_id)

```

```
11-- "Old" syntax using a "CROSS
   JOIN"
12
   SELECT *
13
   FROM author a, book b
14
   WHERE a.author_id = b.author_id
```

OUTER JOIN

OUTER JOIN types help where we want to retain those rows from either the LEFT side or the RIGHT or both (FULL) sides, for which there was no matching row where the predicate yielded true.

A LEFT OUTER JOIN in relational algebra is defined as such:

$$(R \bowtie S) \cup ((R - \pi_{r_1, r_2, \ldots, r_n}(R \bowtie S)) \times \{(\omega, \ldots \omega)\})$$

Or more verbosely in SQL:

```
1 SELECT *
2 FROM author a
3 LEFT JOIN book b USING
  (author_id)
```

This will produce all the authors and their books, but if an author doesn't have any book, we still want to get the author with NULL as their only book value. So, it's the same as writing:

```
1  SELECT *
2  FROM author a
3  JOIN book b USING (author_id)
4
5  UNION
6
7  SELECT a.*, NULL, NULL, NULL, ...,
8  NULL
9  FROM (
10   SELECT a.*
```

```
11  FROM author a
12
13  EXCEPT
14
15  SELECT a.*
16  FROM author a
17  JOIN book b USING (author_id)
    ) a
```

But no one wants to write that much SQL, so OUTER JOIN was implemented.

Conclusion: Say NO to Venn Diagrams

JOINs are relatively easy to understand intuitively. And they're relatively easy to explain using Venn Diagrams. But whenever you do that, remember, that you're making a wrong analogy. A JOIN is not strictly a set operation that can be described with Venn Diagrams. A JOIN is always a cross product with a predicate, and possibly a UNION to add additional rows to the OUTER JOIN result.

So, if in doubt, please use JOIN diagrams rather than Venn Diagrams. They're more accurate and visually more useful.