Cursul 13

OPTIMIZAREA CERERILOR

Ce este

- Limbajul SQL este un limbaj de cereri de nivel inalt, neprocedural.
- Printr-o cerere se specifica **ce** doreste sa obtina utilizatorul si nu si **cum** se ajunge la rezultat.
- Modalitatea de obtinere a acestuia ramane in sarcina sistemului de gestiune care trebuie sa aleaga cea mai putin costisitoare cale.
- Costul este masurat de obicei prin timpul de executie care, asa cum spune si definitia bazei de date din capitolul introductiv, trebuie sa fie rezonabil.

Ce este

- Pentru reducerea timpului de raspuns sistemul efectueaza o serie de modificari ale cererii transformand-o intr-o cerere echivalenta dar care se poate executa mai rapid.
- Aceasta operatie poarta numele de **optimizarea cererii**.

Ce este

- Supuse optimizarii sunt in primul rand cererile de regasire de date (de tip SELECT in SQL).
- Trebuie avut insa in vedere ca si celelalte cereri de tip DML au nevoie de optimizari: pentru a actualiza sau sterge anumite linii dintr-o tabela ele trebuiesc intai localizate iar in cazul inserarilor trebuie verificat daca informatia nu exista deja in tabela.
- In plus o serie de cereri de tip DDL si DCL includ operatii de regasire de date care implicit sunt optimizate de SGBD.

Elemente de cost

- In cazul bazelor de date centralizate (BDC), un aspect foarte important in contextul optimizarii cererilor este minimizarea numarului de accese la disc.
- Sa luam exemplul unui produs cartezian intre doua relatii de cate doua atribute, fie ele R si S, avand **r** respectiv **s** tupluri.
- Fie $\mathbf{n_r}$ respectiv $\mathbf{n_s}$ numarul de tupluri din fiecare relatie care incap intr-un bloc pe disc, si **b** numarul de blocuri disponibile in memoria interna a calculatorului.

Elemente de cost

- Sa presupunem ca procesul de calcul se desfasoara astfel:
- citim numarul maxim posibil de blocuri din R (adica **b-1** blocuri) si
- pentru fiecare inregistrare din R citim in intregime, in ultimul bloc disponibil, relatia S.

Elemente de cost

Atunci:

- Numarul de blocuri citite pentru a parcurge relatia R este: $\frac{r}{n_r}$
- Numarul de parcurgeri al relatiei S este $\frac{r}{(b-1)*n_r}$
- Pentru o parcurgere a relat un numar de blocuri egal cu: $\frac{s}{n_s}$

Elemente de cost

- Deci numarul de accese la disc este in total:

Numarul de blocuri din R + Numarul de parcurgeri ale lui S * Numarul de blocuri din S:

$$\frac{r}{n_r} + \frac{r}{(b-1)*n_r} * \frac{s}{n_s}$$

- care se poate rescrie si ca:

$$\frac{r}{n_r}(1 + \frac{s}{(b-1)*n_s})$$

Elemente de cost

- In cazul relatiilor cu numar mare de tupluri aceasta expresie denota timpi mari pentru executia unei astfel de cereri.
- In cazul de mai sus, deoarece numarul de accese este influentat mai puternic de $\mathbf{r/n_r}$ decat de $\mathbf{s/n_s}$, vom lua ca relatie R pe cea care are acest raport mai mic.

Join

- In implementarea joinurilor, se folosesc de asemenea diverse metode:
  - sortarea uneia dintre relatii dupa atributele implicate in join (in cazul echijoinului)
  - folosirea indecsilor (daca exista) pe campurile implicate in join, pentru acces direct la tuplurile care dau elemente ale rezultatului.
  - micsorarea numarului de tupluri luate in calcul, prin aplicarea mai intai a selectiilor, daca acestea sunt prezente in cerere.

Strategii

- In general, exista o serie de principii (strategii) care duc la micsorarea numarului de accese la disc. Ele sunt urmatoarele (vezi si [Ul 82] - Ullman, J.D. Principles of Database Systems, Second Edition, Computer Science Press, 1982):
- 1. Realizarea selectiilor cit mai devreme posibil.
- 2. Combinarea anumitor selectii cu produse carteziene adiacente pentru a forma un join.

Strategii

- 3. Combinarea secventelor de operatii unare (selectii si proiectii) intr-una singura (o selectie sau/si o proiectie)
- 4. Cautarea subexpresiilor comune, pentru a fi evaluate o singura data.
- 5. Folosirea indecsilor sau sortarea relatiilor, daca se obtine o crestere a performantelor.

Strategii

- 6. Evaluarea diverselor strategii posibile inainte de a incepe procesul de calcul efectiv (in cazul in care sunt posibile mai multe metode de calcul) pentru a alege pe cea mai eficienta.
- Pe baza acestor principii, exista o serie de algoritmi de optimizare pentru evaluarea expresiilor relationale.

Echivalente

- Pentru rescrierea unei expresii in algebra relationala in vederea optimizarii costurilor de evaluare se pot folosi urmatoarele echivalente:
- *1. Comutativitatea joinului si a produsului cartezian:*
- $E1 \times E2 \equiv E2 \times E1$
- $E1 \rhd\lhd E2 \equiv E2 \rhd\lhd E1$
- $E1 \rhd\lhd_F E2 \equiv E2 \rhd\lhd_F E1$

Echivalente

- *2. Asociativitatea produsului cartezian si a joinului:*
- $E1 \times (E2 \times E3) \equiv (E1 \times E2) \times E3$
- $E1 \rhd\lhd (E2 \rhd\lhd E3) \equiv (E1 \rhd\lhd E2) \rhd\lhd E3$
- $E1 \rhd\lhd_{F1} (E2 \rhd\lhd_{F2} E3) \equiv (E1 \rhd\lhd_{F1} E2) \rhd\lhd_{F2} E3$

Echivalente

- *3. Cascada de proiectii:*
- $\pi_{A1...An} (\pi_{B1...Bm}(E)) \equiv \pi_{A1...An} (E)$. Se presupune bineinteles ca $\{Ai\} \subseteq \{Bj\}$
- *4. Cascada de selectii:*
- $\sigma_{F1}(\sigma_{F2}(E)) \equiv \sigma_{F1 \wedge F2} (E)$

Echivalente

- *5. Comutativitatea selectiilor cu proiectiile:*
- Daca F contine doar atribute din A1...An atunci:

$\pi_{A1...An}(\sigma_F (E)) \equiv \sigma_F (\pi_{A1...An}(E))$

- Sau, in cazul general – F contine si alte atribute, B1...Bm - atunci:

$\pi_{A1...An}(\sigma_F (E)) \equiv \pi_{A1...An}(\sigma_F (\pi_{A1...An, B1 ... Bm} (E)))$

Echivalente

- *6. Comutativitatea selectiei cu produsul cartezian:*
- Daca toate atributele din F fac parte din E1:

$\sigma_F (E1 \times E2) \equiv \sigma_F (E1) \times E2$

- Daca $F = F1 \wedge F2$ cu F1 continand doar atribute din E1 si F2 doar din E2:

$\sigma_F (E1 \times E2) \equiv \sigma_{F1} (E1) \times \sigma_{F2} (E2)$

- Daca $F = F1 \wedge F2$ cu F1 continand doar atribute din E1 dar F2 e o expresie generala:

$\sigma_F (E1 \times E2) \equiv \sigma_{F2} (\sigma_{F1} (E1) \times E2)$

Echivalente

- *7. Comutativitatea selectie – reuniune:*

$\sigma_F (E1 \cup E2) \equiv \sigma_F (E1) \cup \sigma_F (E2)$

- *8. Comutativitatea selectie – diferenta:*

$\sigma_F (E1 - E2) \equiv \sigma_F (E1) - \sigma_F (E2)$

Echivalente

- *9. Comutativitatea proiectie – produs cartezian:*
- Daca in lista A1...An atributele B1...Bm sunt din E1 iar C1...Ck din E2 atunci:

$\pi_{A1...An}(E1 \times E2) \equiv \pi_{B1...Bm}(E1) \times \pi_{C1...Ck}(E2)$

- *10. Comutativitatea proiectie – reuniune:*

$\pi_{A1...An}(E1 \cup E2) \equiv \pi_{A1...An}(E1) \cup \pi_{A1...An}(E2)$

Algoritm de optimizare

- In lucrarea [Ul 82] este prezentat un algoritm de optimizare a expresiilor relationale. Acesta este urmatorul:

Intrare: un arbore reprezentand o expresie relationala.

Iesire: program pentru evaluarea expresiei.

Algoritm de optimizare

- **Metoda:**
- ***Pasul 1.*** Fiecare selectie este transformata folosind regula 4 intr-o cascada de selectii:

$\sigma_{F1 \wedge F2 \wedge ... Fn}(E) \equiv \sigma_{F1}(\sigma_{F2}(...(\sigma_{Fn}(E)) ...))$

- ***Pasul 2.*** Fiecare selectie este deplasata in jos folosind regulile 4-8 cât mai aproape de frunze.

Algoritm de optimizare

- ***Pasul 3.*** Folosind regulile 3, 5, 9 si 10, fiecare proiectie este deplasata cât mai jos posibil in arborele sintactic.
- Regula 3 face ca unele proiectii sa dispara, regula 5 sparge o proiectie in doua astfel incat una poate migra spre frunze.
- Daca o proiectie ajunge sa fie dupa toate atributele ea dispare.

Algoritm de optimizare

- ***Pasul 4.*** Cascadele de selectii si proiectii sunt combinate folosind regulile 3-5 intr-o singura selectie, o singura proiectie sau o selectie urmata de o proiectie.
- Aceasta abordare tine de eficienta: e mai putin costisitor sa se faca o singura selectie si apoi o singura proiectie decat daca s-ar alterna de mai multe ori selectii cu proiectii.

Algoritm de optimizare

- ***Pasul 5.*** Nodurile interioare ale arborelui rezultat sunt impartite in grupuri.
- Fiecare nod intern care corespunde unei operatiuni binare devine un grup impreuna cu predecesorii sai imediati cu care sunt asociate operatiuni unare.
- Din bloc fac parte de asemenea orice lant de noduri succesoare asociate cu operatiuni unare si terminate cu o frunza cu exceptia cazului când operatia binara este un produs cartezian neurmat de o selectie cu care sa formeze un echijoin.

Algoritm de optimizare

- ***Pasul 6.*** Se evalueaza fiecare grup astfel incat niciunul nu este evaluat inaintea descendentilor sai.
- Rezulta astfel un program de evaluare a expresiei relationale initiale.

Exemplu

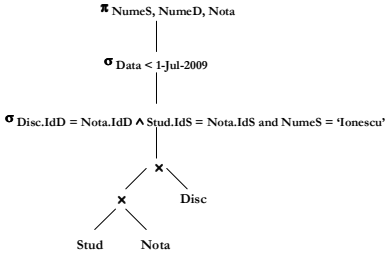
- Exemplu: Fie o baza de date care implementeaza o diagrama entitate asociere formata din 2 entitati (Student, Disciplina) si o asociere binara multi-multi Nota continand notele studentilor si data obtinerii lor ca atribute proprii.
- Structura simplificata a bazei de date este urmatoarea:

```
Stud (IdS, NumeS, ...)
Disc (IdD, NumeD, ...)
Nota (IdS, IdD, Nota, Data)
```


Exemplu

- Fie de asemenea o expresie relationala care afiseaza numele studentului si notele sale pentru studentul cu numele „Ionescu” si note obtinute obtinute pana la 1 iulie 2009:
- π NumeS, NumeD, Nota (σ Data < 1-Jul-2009 (σ Disc.IdD = Nota.IdD $\wedge$ Stud.IdS = Nota.IdS and NumeS = 'Ionescu' ((Stud x Nota) x Disc)))

Exemplu – arbore initial

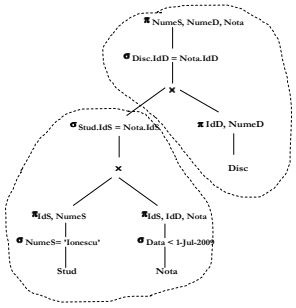


Exemplu

Aplicand algoritmul prezentat:

- Selectia dupa data va ajunge pe ramura tabelei Nota
- Conditia de join se va sparge in trei:
  - o conditie pentru echijoinul Stud cu Nota
  - alta pentru echijoinul rezultatului primului join cu Disc si
  - o conditie dupa numele studentului care coboara pe ramura respectiva.
- Regula 5 generalizata va duce la adaugarea unor proiectii care lasa sa treaca mai departe de la frunze doar atributele necesare operatiilor ulterioare.
- Rezultatul obtinut este urmatorul:

Exemplu – arbore rezultat



Planul rezultat

- Se observa formarea a doua grupuri care se evalueaza in ordinea mentionata in algoritm: intai grupul de jos si apoi grupul de sus care foloseste rezultatul primului grup.
- Pentru fiecare grup produsul cartezian si selectia se pot combina intr-un echijoin.

Studiu de caz: MySQL

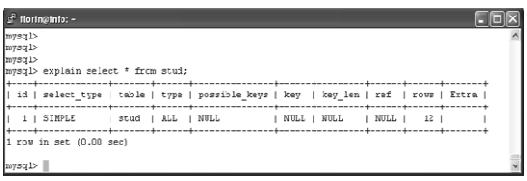
- In cazul MySQL putem obtine informatii despre modul in care sistemul va executa o cerere folosind comanda EXPLAIN.
- In urma examinarii rezultatului acesteia putem decide sa facem schimbari in organizarea datelor (creare de indecsi) sau sa fortam folosirea sau nefolosirea unor indecsi.
- Aceste schimbari au ca scop cresterea vitezei de evaluare a cererii.

EXPLAIN

- EXPLAIN este folosit pentru a obtine informatii despre modul in care serverul executa o cerere.
- Daca EXPLAIN precede o cerere SELECT vor fi afisate informatii furnizate de optimizatorul de cereri despre planul de executie al regasirii respective, inclusiv despre modul in care se efectueaza eventualele joinuri si in ce ordine.

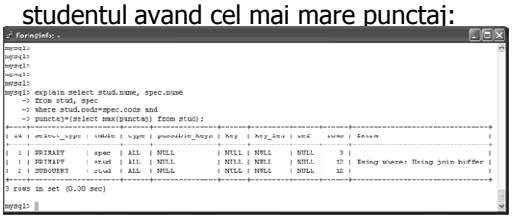
Exemplul 1

- Un prim exemplu: o cerere pe o tabela, fara subcereri:



Exemplul 2

- Un alt exemplu: cerere cu join continand o subcerere. Cererea returneaza numele studentului si numele specializarii pentru studentul avand cel mai mare punctaj:



Coloana id

- Coloanele rezultatului comenzii EXPLAIN sunt urmatoarele:
- **Id**: Identificatorul cererii **SELECT.**
- In Exemplul 2 sunt 2 cereri SELECT, cea principala si subcererea care returneaza maximul.
- In rezultat sunt 3 linii deparece sunt 3 parcurgeri de tabela: doua in cererea principala (care e un join) si una in subcerere.

Coloana select_type

- **select_type**: Tipul cererii SELECT. In MySQL 5.5 avem urmatoarele tipuri:
  - SIMPLE: SELECT fara UNION ori subcereri
  - PRIMARY: SELECT exterior (cererea principala)
  - UNION: Al doilea sau urmatoarele SELECT dintr-un UNION

Coloana select_type

- DEPENDENT UNION: Al doilea sau urmatoarele SELECT dintr-un UNION, dependent de cererea de nivel superior
- UNION RESULT: Resultatul unui UNION.

Coloana select_type

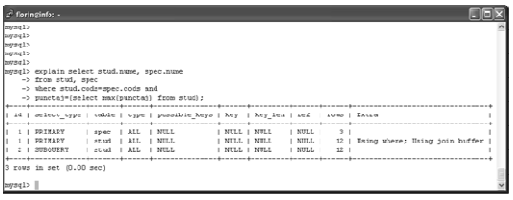
- SUBQUERY: Primul SELECT din subcerere
- DEPENDENT SUBQUERY: Primul SELECT din subcerere, dependent de cererea de nivel superior (se evalueaza pentru fiecare set de valori distinct provenit din aceasta)

Coloana select_type

- DERIVED: SELECT in subcerere pe clauza FROM
- UNCACHEABLE SUBQUERY: O subcerere al carei rezultat trebuie reevaluat pentru fiecare linie din cererea de nivel superior

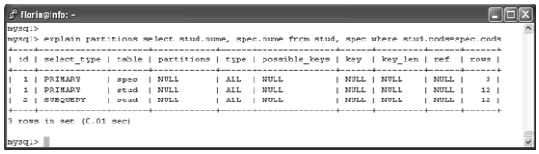
Coloana table

- Tabela parcursa. In cazul unui join sunt mai multe tabele si vom avea in rezultat cate o linie pentru fiecare parcurgere.



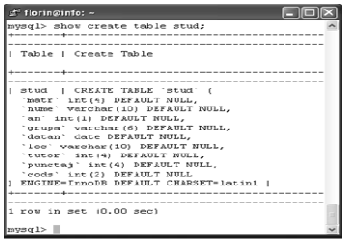
Coloana partitions

- **partitions**: Doar cand se foloseste clauza PARTITIONS: partitia din care vor fi luate liniile. Aceste partitii se speciifica la crearea tabelei (daca e cazul).



Coloana partitions

- Pentru a vedea eventualele partitii pentru o tabela existenta se poate folosi comanda SHOW CREATE TABLE:



Coloana type

- **type:** Tipul de join. ALL semnifica 'full table scan' (parcurgerea tuturor liniilor).
- Exista mai multe valori pentru aceasta coloana:
- **system**: tabela are o singura linie = e tabela de sistem

Coloana type

- **const**: tabela are o singura linie care corespunde rezultatului cererii, linie care e citita la inceput iar valorile respective sunt ca niste constante.
- Apare cand se compara o cheie unica sau primara cu o valoare constanta.

```
SELECT * FROM tbl_name
WHERE primary_key=1;
SELECT * FROM tbl_name
WHERE primary_key_part1=1 AND
  primary_key_part2=2;
```


Coloana type

- **eq_ref**: din tabela se citeste o singura linie pentru fiecare combinatie de linii din tabelele precedente.
- Apare de exemplu atunci cand se compara pentru egalitate coloane indexate.
- Ca si la system si const evaluarea joinului e foarte rapida.

Coloana type

- **Exemplu caz eq_ref:**

```
SELECT * FROM ref_table,other_table
WHERE ref_table.key_column =
  other_table.colum;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1 =
  other_table.column  AND
  ref_table.key_column_part2 = 1;
```

Coloana type

- **ref**: sunt citite din tabela toate liniile cu aceeasi valoare de index pentru fiecare combinatie de linii din tabelele precedente.
- De exemplu atunci cand se face un join care foloseste doar partea stanga a unei chei multiple (care identifica mai multe linii si nu una) sau daca e vorba de un index ne-unic.

Coloana type

- Se foloseste un astfel de join pentru comparatii cu = si <=> (NULL safe equal) pe coloane indexate.
- NULL safe equal - exemple:

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
```

Returneaza: `1, 1, 0 mysql>`

```
SELECT 1 = 1, NULL = NULL, 1 = NULL;
```

Returneaza: `1, NULL, NULL`

Coloana type

- Exemplu caz ref (key_column e un index ne-unic):

```
SELECT * FROM ref_table WHERE
  key_column=expr;
SELECT * FROM ref_table,other_table
  WHERE
  ref_table.key_column=other_table.column;
SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1 =
  other_table.column
  AND ref_table.key_column_part2 = 1;
```


Coloana type

- **fulltext**: join folosind un index de tip FULLTEXT (la regasire, in loc de LIKE se poate folosi MATCH(…)…AGAINST(…).
- **ref_or_null**: este ca ref dar se face si o cautare pentru linii care contin NULL. Este folosit pentru optimizarea lui IS NULL sau in subcereri.

```
SELECT * FROM ref_table
WHERE key_column = expr OR key_column IS
  NULL;
```


Coloana type

- **index_merge**: se foloseste optimizarea de tip 'index merge'.
- Se foloseste cand rezultatul cererii se poate obtine prin unirea mai multor rezultate partiale provenite din parcurgeri de tip **range** (descrise peste cateva slide-uri).
- In coloana **key** din rezultatul EXPLAIN se spune care sunt indecsii folositi (vezi mai jos).

Coloana type

- Exemple de cereri de tip **index_merge**:

```
SELECT * FROM tbl_name WHERE key1 = 10 OR
  key2 = 20;
SELECT * FROM tbl_name WHERE (key1 = 10
  OR key2 = 20) AND non_key=30;
SELECT * FROM t1, t2 WHERE (t1.key1 IN
  (1,2) OR t1.key2 LIKE 'value%') AND
  t2.key1=t1.some_col;
SELECT * FROM t1, t2 WHERE t1.key1=1 AND
  (t2.key1=t1.some_col OR
  t2.key2=t1.some_col2);
```

Coloana type

- **unique_subquery**: este folosit in loc de ref pentru cereri cu IN. Subcererea e inlocuita cu o cautare in index (cheia primara are automat unul):

```
value IN (SELECT primary_key FROM
  single_table WHERE some_expr)
```

- **index_subquery**: ca si unique_subquery, dar pentru indecsi non-unici:

```
value IN (SELECT key_column FROM
  single_table WHERE some_expr)
```


Coloana type

- **range**: doar liniile care au valori intr-o anumita plaja sunt regasite, utilizand un index.
  - In coloana **key** din rezultatul EXPLAIN se spune care este acesta.
  - E folosit la joinuri cu operatii de comparatie, IS NULL, BETWEEN sau IN.
  - Coloana **ref** din rezultatul EXPLAIN este NULL in acest caz.

Coloana type

```
SELECT * FROM tbl_name  WHERE
  key_column = 10;
SELECT * FROM tbl_name  WHERE
  key_column BETWEEN 10 and 20;
SELECT * FROM tbl_name  WHERE
  key_column IN (10,20,30);
SELECT * FROM tbl_name  WHERE
  key_part1 = 10 AND key_part2 IN
  (10,20,30);
```


Coloana type

- **index**: este la fel ca ALL dar se parcurge indexul nu datele.
- **ALL**: o parcurgere completa a tabelei este efectuata pentru fiecare combinatie de linii din tabelele precedente.
  - O astfel de strategie nu este de dorit in cazul in care tabela are mai multe linii.
  - Pentru a impiedica acest mod de calcul al joinului se pot adauga indecsi care sa creasca viteza.

Coloana possible_keys

- **possible_keys**: Indecsi posibil de utilizat.
- Are valoarea NULL daca nu exista indecsi relevanti.
- Executia se poate imbunatati in acest caz prin crearea de indecsi pe coloanele folosite in WHERE.
- Indecsii unei tabele se pot vizualiza cu **SHOW INDEX FROM nume_tabela;**

Coloana key

- **key**: Indexul ales. E posibil sa nu fie dintre cele de mai sus ci unul care contine toate coloanele regasite (parcurgerea tabelei se face prin index - index scan).
- **Observatie:** Pentru a forta folosirea unui index sau nefolosirea lui se pot folosi clauzele FORCE INDEX, USE INDEX sau IGNORE INDEX.

Coloana key_len

- **key_len**: lungimea cheii alese. Poate arata cate parti ale unui index multiplu sunt folosite; NULL in cazul in care coloana **key** e NULL

Coloanele ref si rows

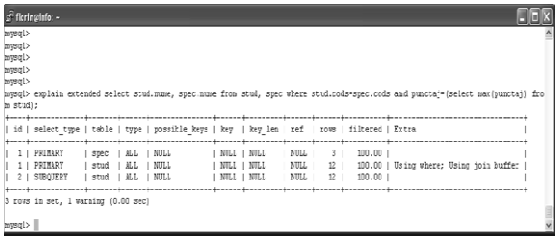
- **ref**: aici sunt listate coloanele sau constantele comparate cu indexul al carui nume este pe coloana key.
- **rows**: Numarul estimate de linii examinate. Pentru stocarea de tip InnoDB acest numar este o estimare si poate sa nu fie intotdeauna exact.

Coloanele filtered si extra

- **filtered**: Procentul de linii indicate de conditia pe acea tabela. Apare doar la EXPLAIN EXTENDED
- **extra**: Informatii aditionale (vezi documentatia indicata mai jos).
- Mai multe informatii despre coloanele rezultatului pot fi gasite la adresa: http://dev.mysql.com/doc/refman/5.5/en/explain-output.html

EXPLAIN EXTENDED

- In acest caz rezultatul include si coloana filtered:



Exemple prezentare MySQL

- Documentatia MySQL

http://dev.mysql.com/doc/refman/5.5/en/using-explain.html

- Carte: Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, High Performance MySQL, 3rd Edition, O'Reilly Media, 2012

http://www.it-ebooks.info/book/676/

- Se recomanda si parcurgerea prezentarii:

http://www.percona.com/files/presentations/percona-live/dc-2012/PLDC2012-mysql-query-optimization.pdf

Sfarsitul cursului 13