Curs 10

2011/2012

Tehnici moderne de proiectare a aplicatiilor web

Teme de proiect

- Functionalitate
  - La toate temele **1p** din nota este obtinut de indeplinirea functionalitatii cerute.
  - orice tehnologie, orice metoda, "sa faca ceea ce trebuie"
- Forma paginii prezinta importanta
  - dependenta de dificultatea temei
- Initiativa
  - **Necesitatea** investigarii posibilitatilor de imbunatatire
- Cooperare
  - Necesitatea conlucrarii intre 2 studenti cu doua teme "pereche"

Curs 9

- Adaptarea aplicatiei de magazin virtual pentru lucrul cu baze de date MySql
- Planul aplicatiei poate fi pastrat
- Activitate suplimentara
  - termen limita: S14 inainte de curs
- Proiect
  - in mare masura decide nota finala
  - **cea mai importanta proba**
    - curs **SI** laborator – suport pentru crearea aplicatiei la proiect
  - termen limita: S14, laborator

Recompensa activitate suplimentara

- Raspunsul corect va fi recompensat cu:
  - **2p** in plus la nota de laborator (se pot compensa astfel eventuale absente)
  - **2p** in plus la nota de la testarea finala (examen)
- Nota de la proiect
  - Nu este influentata
- Nota finala se obtine prin medie ponderata **dupa** aplicarea suplimentelor amintite mai sus

Regulament recompensa

- Raspunsul si codul de corectie trebuie trimise individual prin email
- Codul trebuie sa fie functional
- Maxim **2** incercari pentru fiecare student
- Studentii pot discuta intre ei **dar**
- Oricare **doua raspunsuri identice se elimina reciproc**

Curs 9

I.	HTML si XHTML (recapitulare)	1 oră
II	CSS	2 ore
III	Baze de date, punct de vedere practic	1 oră
IV	Limbajul de interogare SQL	4 ore
V	PHP - HyperText Preprocessor	8 ore
VI	XML - Extended Mark-up Language si aplicatii	4 ore
VII	Conlucrare intre PHP/MySql, PHP/XML, Javascript/HTML	2 ore
VIII	Exemple de aplicatii	6 ore
	Total	28 ore

Relatii in Bazele de date

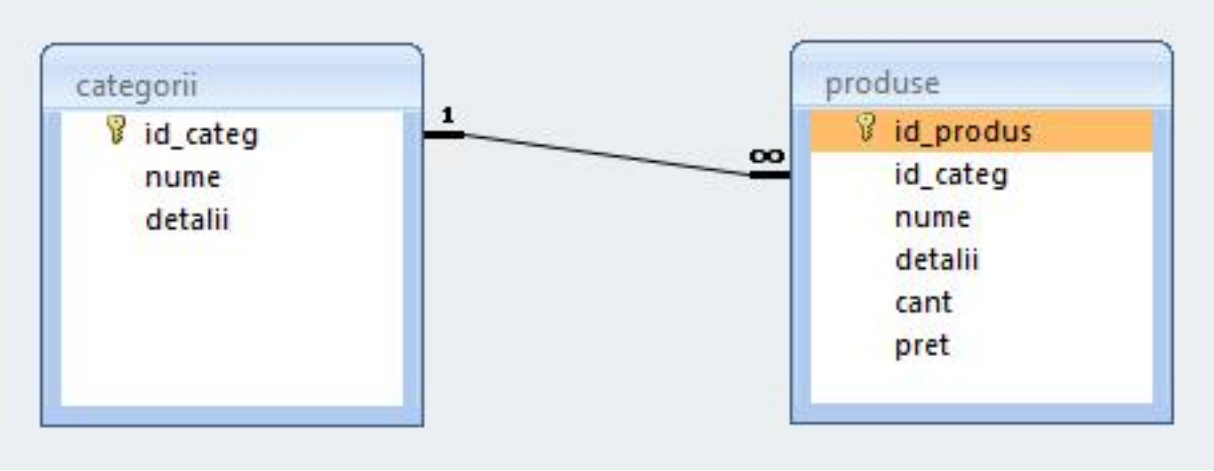
- Respectarea formelor normale ale bazelor de date aduce nenumarate avantaje
- Efectul secundar este dat de necesitatea separarii datelor intre mai multe tabele
- In exemplul utilizat avem doua concepte diferite din punct de vedere logic
  - produs
  - categorie de produs

Relatii in Bazele de date

- In exemplul utilizat avem doua concepte diferite din punct de vedere logic
  - produs
  - categorie de produs
- Cele doua tabele nu sunt independente
- Intre ele exista o legatura data de functionalitatea dorita pentru aplicatie: **un produs va apartine unei anumite categorii de produse**

Relatii in Bazele de date

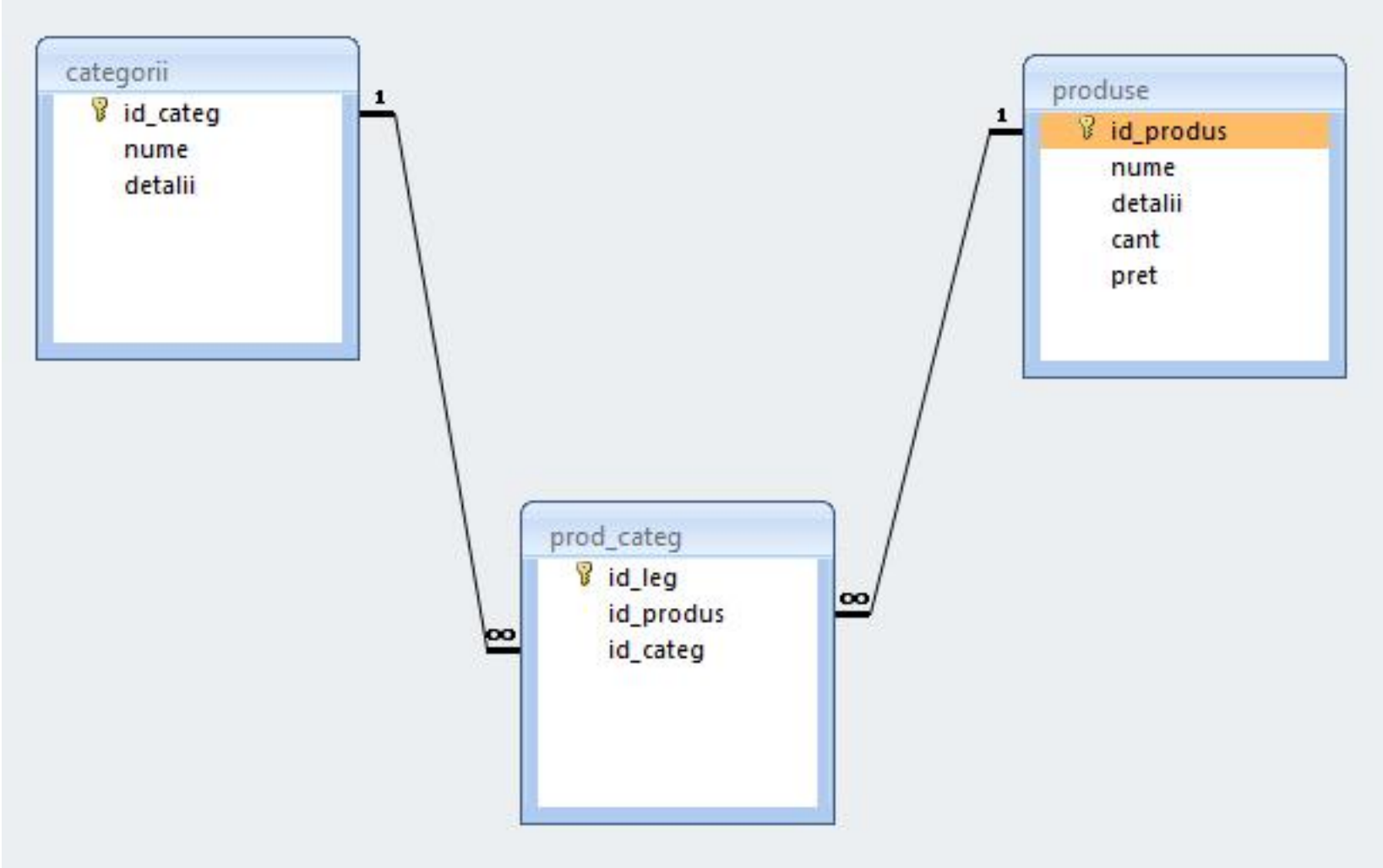
- Legaturile implementata
  - One to Many
  - in tabelul “produse” apare cheia externa (foreign key): “id_categ”



Relatii in Bazele de date

- Daca se doreste o situatie cand un produs poate apartine **mai multor categorii** (o carte cu CD poate fi inclusa si in “papetarie” si in “audio-video”)
  - relatia devine de tipul **Many to Many**
  - e necesara introducerea unui tabel de legatura cu coloanele “id_leg” (cheie primara), “id_categorie” si “id_produs” (chei externe)

Relatii in Bazele de date



Limbaj SQL

MySql – eficienta

- eficienta unei aplicatii MySql
  - 25% **alegerea corecta a tipurilor de date**
  - 50% **crearea indecsilor necesari in aplicatii**
  - 20% **cresterea complexitatii interogarilor pentru a "muta" prelucrarile pe server-ul de baze de date**
  - 5% **scrierea corecta a interogarilor**

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 cuvantului 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_nume_lung 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)

Metode de stocare

- Metoda de stocare a datelor nu e o caracteristica a server-ului ci a fiecarui tabel in parte
- Exemplu ulterior CREATE: "ENGINE = InnoDB"
- MySql suporta diferite metode de stocare, fiecare cu avantajele/dezavantajele sale
- Implicit se foloseste metoda MyISAM, dar la instalarea server-ului (laborator 1) o anumita selectie poate schimba valoarea implicita in InnoDB
- **Alegerea metodei de stocare potrivita are implicatii majore asupra performantei aplicatiei**

Metode de stocare

- MyISAM
- InnoDB
- Memory
- Merge
- Archive
- Federated
- NDBCLUSTER
- CSV
- Blackhole
- Example

Metode de stocare

- **MyISAM**
  - metoda de stocare implicita in MySql
  - performanta ridicata (resurse ocupate si viteza)
  - posibilitatea cautarii in intregul text (index FULLTEXT)
  - blocare acces la nivel de tabel
  - **nu** accepta tranzactii
  - **nu** accepta FOREIGN KEY
    - probleme relative la integritatea datelor
- **InnoDB**
- **Memory**

Metode de stocare

- MyISAM
- InnoDB
  - devine metoda de stocare implicita in MySql daca la instalare se alege model tranzactional
  - performanta medie (resurse ocupate si viteza)
  - blocare acces la nivel de linie
  - nu accepta index FULLTEXT
  - accepta tranzactii
  - accepta FOREIGN KEY
    - probleme mai putine la integritatea datelor prin constrangeri intre tabele
- Memory

Metode de stocare

- MyISAM
- InnoDB
- Memory
  - metoda de stocare recomandata pentru tabele temporare
  - performanta maxima (viteza – datele sunt stocate in RAM)
    - **la oprirea server-ului datele se pierd**, tabelul este pastrat dar va fi fara nici o linie
  - **nu** accepta tipuri de date mari (BLOB, TEXT) – maxim 255 octeti
  - **nu** accepta index FULLTEXT
  - **nu** accepta tranzactii
  - **nu** accepta FOREIGN KEY
    - probleme relative la integritatea datelor

Interogari SQL

Interogari

- Interogarile 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 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**

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 aspra 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
- Fisierele 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 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`)
     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
- Fisierele SQL “backup” contin succesiunea DROP..., CREATE... pentru a inlocui datele in intregime

Interogari SQL

Interogari

- Interogarile 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
  - poate 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 acepta functii aggregate, dar se aplica la sfarsit, exact inainte de a fi trimise datele clintului, **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 cuvantului 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 conditile 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 MxN 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 SELECT * FROM clienti NATURAL JOIN comenzi;
```

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 "subinterigare" 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);

Laborator 2

- Se recomanda aplicarea din nou a exercitiilor din laboratorul 2 pentru exemple de interogari, JOIN, subquery, JOIN cu subquery

Contact

- Laboratorul de microunde si optoelectronica
- http://rf-opto.etti.tuiasi.ro
- rdamian@etti.tuiasi.ro