

# **Programare Web**

## **Curs 6**

# **Capitolul 5. Acces la baze de date din PHP**

# Obiectiv

- ◆ Obiectivul acestui capitol este prezentarea modului in care se poate face accesul la baze de date din PHP.
- ◆ Vor fi prezentate functiile de acces pentru MySQL si Oracle.
- ◆ Functiile ODBC sunt similare celor din MySQL si Oracle.

# Caracteristici

- ◆ Pachetul PHP pune la dispozitie functii pentru accesul la o multitudine de SGBD-uri si pachete de gestiune a datelor
- ◆ Din pacate aceste functii nu sunt standardizate ci de obicei copiaza functii existente in API-urile pachetelor respective.
- ◆ Din aceasta cauza pentru fiecare tip de SGBD exista un set diferit de functii care permit accesul la datele din baza de date

# Caracteristici

- ◆ Pasii in cazul regasirii datelor sunt in general urmatorii:
  1. Stabilirea unei conexiuni cu serverul de baze de date (cine se conecteaza?)
  2. Trimiterea spre executie a cererii de tip SELECT
  3. Preluarea linie cu linie a datelor din tabela rezultat si prelucrarea acestora
  4. Eliberarea resurselor ocupate de rezultat
  5. Inchiderea conexiunii, dupa executia repetata a pasilor 2-4

# Caracteristici

- ◆ In cazul executiei altor tipuri de cereri (insert, update, delete, cereri DDL, etc) se executa pasii 1, 2 si 5, neexistand o tabela rezultat
- ◆ In toate cazurile, la aparitia unei erori se pot obtine informatii despre ea folosind functii care intorc textul mesajului de eroare.

# Functii MySQL - conectare

1. Conectare: Pentru conectarea la SGBD exista mai multe functii care pot fi folosite alternativ:

```
resource mysql_connect(string [hostname] [:port]  
    , string [username] , string [password], bool  
    [new_link], int [client_flags] );
```

- ◆ Întoarce un identificator de conexiune (numeric, pozitiv) în caz de succes și *false* în caz de eșec.
- ◆ Toate argumentele sunt opționale, valorile implicite fiind *localhost:3306*, numele utilizatorului care deține procesul server și un șir vid pentru parolă.

# Funcții MySQL - conectare

- ◆ New\_link forțează returnarea unui nou descriptor de conexiune chiar dacă există deja o conexiune (implicit nu) cu acel server iar client\_flags specifică o serie de alte setări (implicit e ignorat)
- ◆ În cazul unui apel cu aceleași argumente cu ale unei conexiuni deja deschise va întoarce identificatorul acesteia.
- ◆ Pot exista simultan mai multe conexiuni deschise cu unul sau mai multe servere. Conexiunile se închid cu *mysql\_close()* sau automat la terminarea scriptului.
- ◆ Exemplu:

```
$conexiune = mysql_connect("localhost");
```

# Funcții MySQL - conectare

```
resource mysql_pconnect(string [hostname] [:port]  
    , string [username] , string [password] ],  
    int [client_flags] );
```

- ◆ Întoarce un identificador de conexiune persistentă sau *false* în caz de eșec.
- ◆ Se aseamănă cu *mysql\_connect()* cu deosebirea că o astfel de conexiune nu se închide la terminarea scriptului și nici la executia *mysql\_close()*.
- ◆ Conexiunile persistente rămân deschise urmând să fie returnate ca rezultat când se încearcă deschiderea de noi conexiuni cu aceiași parametri.



# Funcții MySQL- ce BD

## 2. Specificarea bazei de date:

```
bool mysql_select_db(string database_name,  
                    resource [id_conexiune] );
```

- ◆ Specifică numele bazei de date care va fi exploatată prin acea conexiune (serverul MySQL poate gestiona mai multe baze de date).
- ◆ Întoarce *true* pentru succes și *false* pentru eșec.
- ◆ Toate cererile care vor fi trimise în viitor cu *mysql\_query()* se vor executa în baza de date specificată.
- ◆ Exemplu:

```
mysql_select_db("test_students",  
              $conexiune );
```

# Funcții MySQL - cereri

## 1. Execuția unei cereri SQL:

```
resource mysql_query(string cerere,  
resource [id_conexiune] );
```

- ◆ Efectul este trimiterea cererii către server pentru a fi procesată.
- ◆ Întoarce: *true/resursa* în caz de reușită, *false* în caz de eșec.
- ◆ În cazul în care cererea a fost un SELECT rezultatul întors este o resursa = identificator de rezultat.
- ◆ Spațiul ocupat de un rezultat poate fi eliberat ulterior cu *mysql\_free\_result()*.
- ◆ Exemplu:

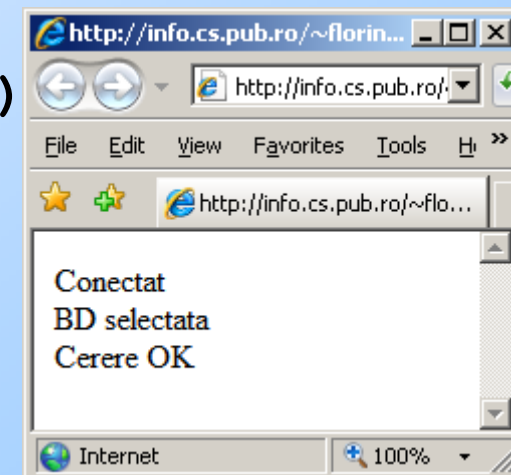
```
$rezultat = mysql_query("select * from  
studenti", $conexiune );
```

# Funcții MySQL - cereri

- ◆ Observație: în afara de funcția `mysql_connect` toate celelalte funcții `mysql` care utilizează un descriptor de conexiune folosesc în cazul lipsei acestui parametru ultima conexiune deschisă.
- ◆ Deci dacă într-o pagină nu se folosește decât o singură conexiune MySQL atunci nu este necesară stocarea și folosirea ei explicită în funcțiile utilizate.
- ◆ Iată un exemplu:

# Funcții MySQL - cereri

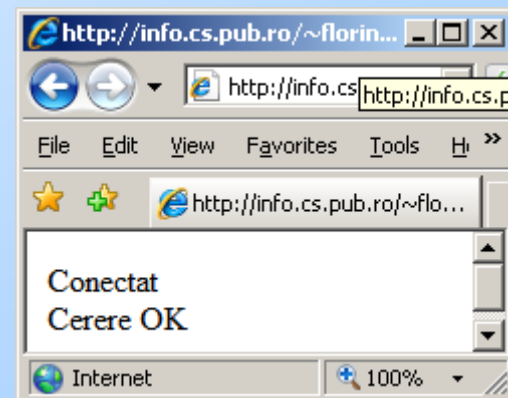
```
<?php
mysql_connect("localhost", "stud", "studpw") or
    die('Conexiune esuata');
echo "Conectat<br>";
// nu folosim descriptor de conexiune:
mysql_select_db("stud") or die ('Selectie BD
    esuata');
echo " BD selectata<br>";
// si aici aici:
mysql_query("select * from produse")
    or die('cerere esuata');
echo " Cerere OK<br>";
?>
```



# Funcții MySQL - cereri

- ◆ Exista și funcția `mysql_db_query`:  
`resource mysql_db_query(string database, string cerere,  
resource [id_conexiune] );`
- ◆ Folosirea acestei funcții nu este recomandată de situl php (deprecated).
- ◆ Efectul ei este cel cumulat al `mysql_select_db` și `mysql_query`
- ◆ Exemplu:

```
<?php  
mysql_connect("localhost", "stud", "studpw") or  
die('Conexiune esuata');  
echo "Conectat<br>";  
mysql_db_query("stud",  
"select * from produse")  
or die('Cerere esuata');  
echo " Cerere OK<br>";  
?>
```



# Funcții MySQL - erori

Detectarea erorii se face cu funcțiile:

```
int mysql_errno(int [id_conexiune] );
```

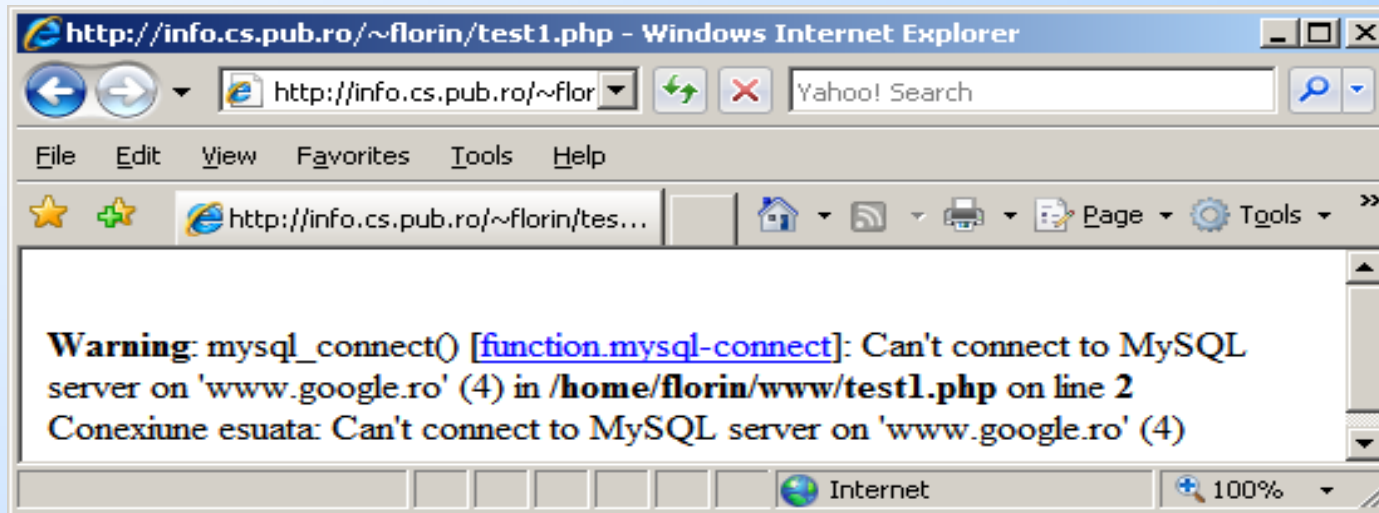
- ◆ Întoarce codul de eroare pentru operația precedentă pe acea conexiune (implicit ultima deschisă, ca mai înainte).

```
string mysql_error(int [id_conexiune] );
```

- ◆ Întoarce textul mesajului de eroare pentru operația precedentă pe acea conexiune.

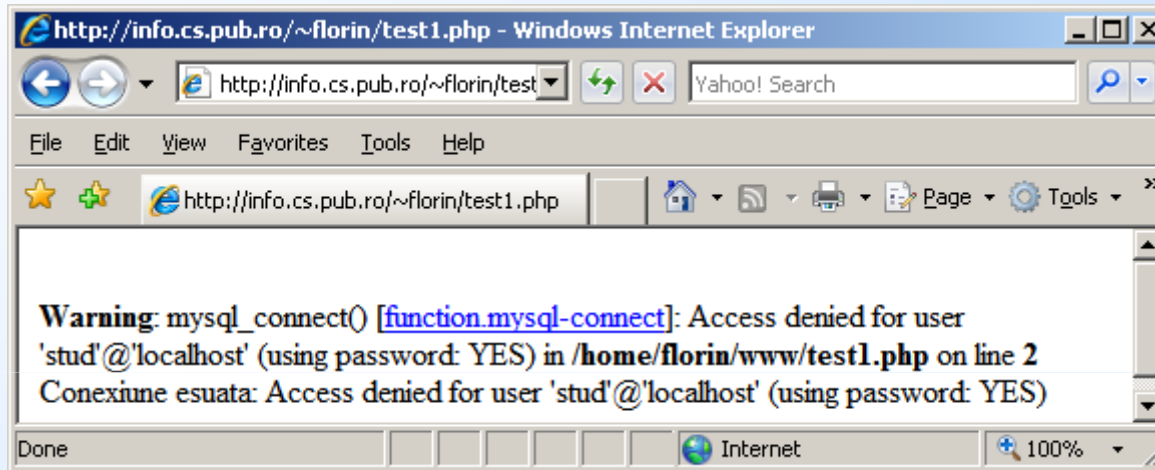
# Funcții MySQL - erori

- ◆ Aceste erori sunt returnate de serverul MySQL.
- ◆ Exceptie face incercarea unei conexiuni cu un host unde nu exista server MySQL.
- ◆ In acest caz mesajul va fi ca mai jos:

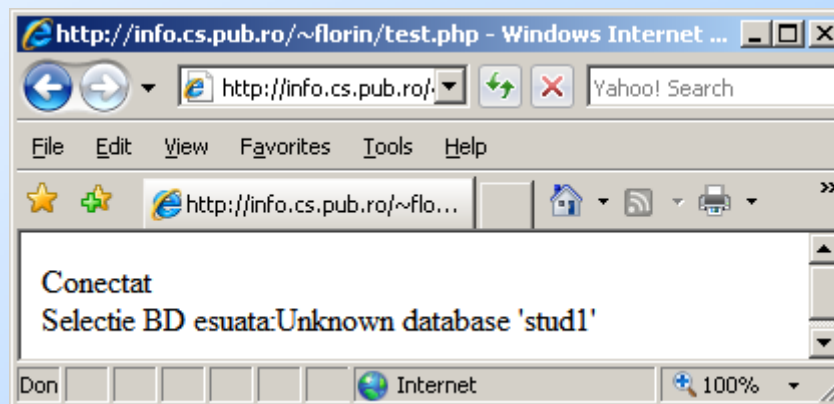


# Exemple

- ◆ Parola gresita la conectare:



- ◆ Nume baza de date eronat

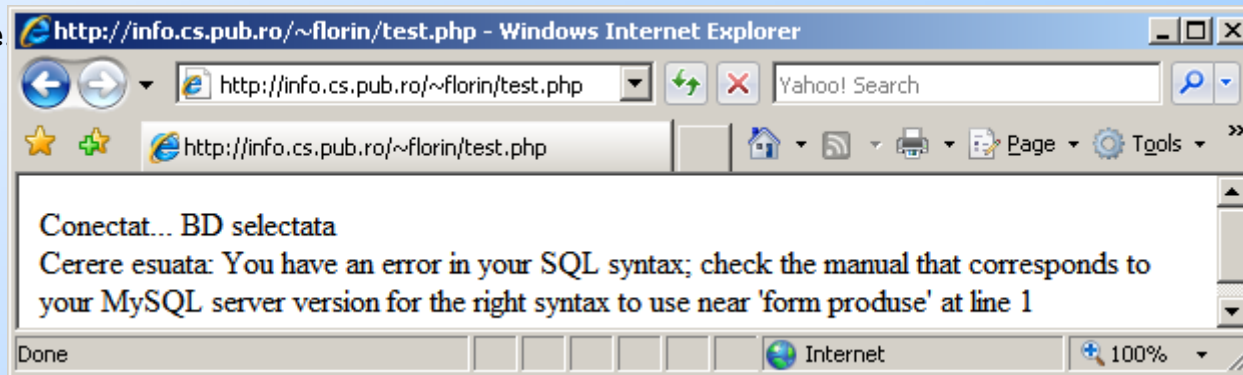




# Exemple

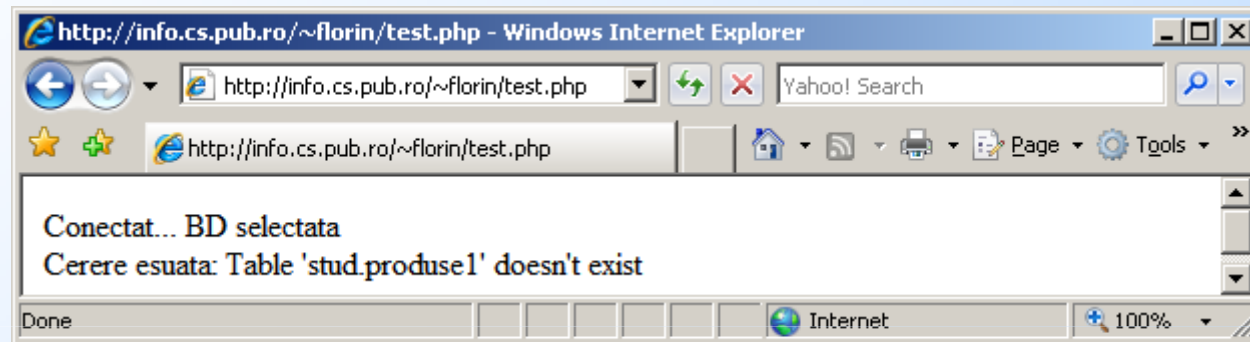
## ◆ Eroare de sintaxa in cerere:

```
<?php
mysql_connect("localhost", "stud", "studpw")
    or die('Conexiune esuata');
echo "Conectat...";
mysql_select_db("stud")
    or die ('Selectie BD esuata: '.mysql_error());
echo " BD selectata<br>";
mysql_query("select * form produse")
    or die('Cerere esuata: '.mysql_error());
echo " Cere
?>
```

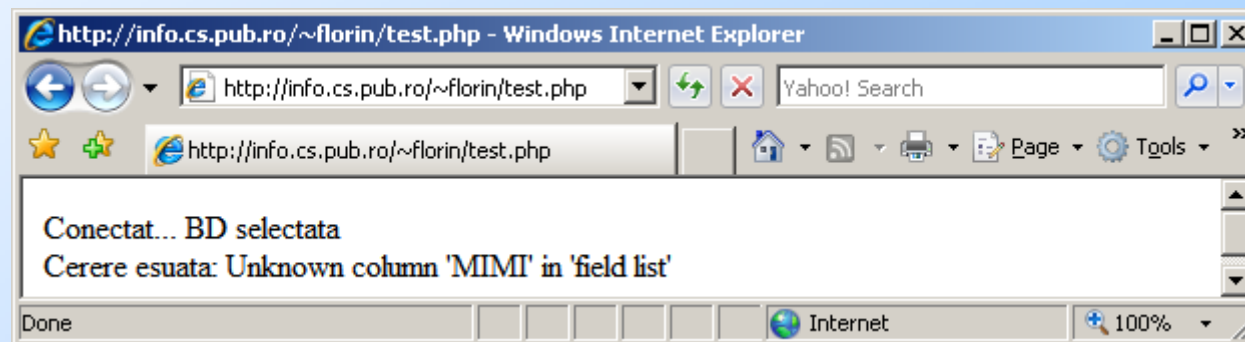


# Exemple

## ◆ Tabela inexistentă:



## ◆ Coloana inexistentă:



# Funcții MySQL - Fetch

- ◆ Incarcarea unei linii din rezultatul intors de o cerere SELECT se poate face folosind una dintre functiile urmatoare:
  1. `mysql_fetch_row()` – Intoarce o linie de rezultat ca tablou cu indici numerici
  2. `mysql_fetch_array()` - Intoarce o linie de rezultat ca tablou ce poate fi folosit atat cu indici numerici cat si ca tablou asociativ
  3. `mysql_fetch_assoc()` - Intoarce o linie de rezultat ca tablou asociativ
  4. `mysql_fetch_object()` - Intoarce o linie de rezultat ca obiect

# Funcții MySQL - Fetch

```
array mysql_fetch_row(resource rezultat);
```

- ◆ Întoarce un array (neasociativ, accesat prin indici) cu valorile următoarei linii din rezultat sau *false* dacă nu mai sunt linii.

- ◆ Exemplu:

```
while($linie = mysql_fetch_array($rezultat))  
{  
    echo $linie[0];  
    echo $linie[1];  
    echo $linie[2]; }  
}
```

# Funcții MySQL - Fetch

```
array mysql_fetch_array(resource  
    rezultat, int [tip_rezultat]);
```

- ◆ Încarcă următoarea linie din rezultat într-un *array* asociativ.
- ◆ În caz de eroare întoarce *false*.
- ◆ Este asemănătoare cu *mysql\_fetch\_row()* dar elementele liniei pot fi accesate și după numele coloanei din rezultat.
- ◆ În cazul în care mai multe coloane au același nume (pentru un join de exemplu) ultima coloană cu acel nume va avea prioritate, celelalte putând fi accesate prin indici.

# Functii MySQL - Fetch

- ◆ Tipul rezultatului (parametru optional) poate fi `MYSQL_ASSOC`, `MYSQL_NUM` sau valoarea implicita `MYSQL_BOTH`. Se poate astfel ca tabloul sa fie accesat doar ca asociativ sau doar cu indici numerici
- ◆ Exemplu:

```
while($linie = mysql_fetch_array($rezultat))  
{  
    echo $linie["nume"]; }  
}
```

# Funcții MySQL - Fetch

```
array mysql_fetch_asoc(resource  
rezultat);
```

- ◆ Încarcă următoarea linie din rezultat într-un *array* asociativ.
- ◆ În caz de eroare întoarce *false*.
- ◆ Este identic cu `mysql_fetch_array` având parametrul `tip_rezultat` egal cu `MYSQL_ASSOC`
- ◆ Exemplu:

```
while($linie = mysql_fetch_array($rezultat))  
{  
    echo $linie["nume"]; }  
}
```

# Funcții MySQL - Fetch

```
array mysql_fetch_object(resource rezultat[,  
    string nume_clasa [, array $parametri ] ] );
```

- ◆ Încarcă următoarea linie din rezultat într-un *obiect*.
- ◆ În caz de eroare întoarce *false*.
- ◆ Nume\_clasa specifica numele clasei (implicit stdClass)
- ◆ Parametrii formeaza un array care va fi transmis constructorului pentru clasa nume\_clasa
- ◆ Nu se mai pot in acest caz accesa elementele inregistrarii incarcate prin indici numerici ci doar prin numele lor.



# Exemplu

```
<?php
mysql_connect ("localhost", "stud", "stu
    or die ('Conexiune esuata');
echo "Conectat...";
// nu folosim descriptor de conexiune l
mysql_select_db ("stud")
    or die ('Selectie BD esuata: '.mysql_
echo " BD selectata<br>";
// si nici la query
$resultat = mysql_query ("select * from
    or die ('Cerere esuata: '.mysql_error (
echo " Cerere OK<br>";
while ($linie=mysql_fetch_object ($resultat))
{   echo $linie->codp;
    echo $linie->nume, "<br>";
}
?>
```



# Funcții MySQL - Fetch

- ◆ La încărcarea unei linii de rezultat, pointerul intern în rezultat avansează cu 1. El se poate 'misca' și altfel folosind funcția

```
int mysql_data_seek(resource id_rezultat, int  
                    numar_linie);
```

- ◆ Linia specificată devine linia curentă în cadrul rezultatului: următorul apel *mysql\_fetch\_???*() va întoarce această linie.
- ◆ Întoarce: *true* în caz de reușită, *false* în caz de eșec.
- ◆ Nota: Pentru prima linie folosim numărul 0!
- ◆ Exemplu:

```
int mysql_data_seek($rezultat, $j);
```

# Functii MySQL - Informatii

```
int mysql_num_fields(resource rezultat);
```

- ◆ Întoarce numărul de câmpuri (coloane) dintr-un rezultat de cerere SELECT.

```
int mysql_num_rows(resource rezultat);
```

- ◆ Întoarce numărul de linii dintr-un rezultat de cerere SELECT.

```
object mysql_fetch_field(resource rezultat, int  
[offset_camp] );
```

- ◆ Întoarce un obiect conținând informații despre un câmp (coloană) al rezultatului. Dacă offsetul de câmp nu este specificat este considerat următorul câmp al rezultatului (se poate seta cu *mysql\_field\_seek()*). Porneste de la 0!

# Functii MySQL - Informatii

Proprietățile obiectului întors de `mysql_fetch_field` sunt:

- ◆ *name* - nume coloană
- ◆ *table* - numele tabelului din care provine acea coloană
- ◆ *def* - valoarea implicată a coloanei
- ◆ *max\_length* - lungimea maximă pentru acea coloană
- ◆ *not\_null* - 1 dacă acea coloană nu poate conține valori nule.
- ◆ *primary\_key* - 1 dacă acea coloană este o cheie primară.
- ◆ *unique\_key* - 1 dacă acea coloană este cheie unică
- ◆ *multiple\_key* - 1 dacă acea coloană este o cheie neunică
- ◆ *numeric* - 1 pentru coloană numerică
- ◆ *blob* - 1 pentru coloană de tip BLOB
- ◆ *type* - tipul coloanei
- ◆ *unsigned* - 1 pentru coloane unsigned
- ◆ *zerofill* - 1 pentru coloane zero-filled

# Functii MySQL - Informatii

```
string mysql_field_name(resource rezultat, int  
                        numar_camp);
```

- ◆ Întoarce numele câmpului cu numărul specificat dintr-un rezultat (porneste de la 0!).

- ◆ Exemplu:

```
$numecamp = mysql_field_name($rezultat, $j);
```

```
string mysql_field_type(resource rezultat, int  
                        offset_camp);
```

- ◆ Este similar cu *mysql\_field\_name()* întorcând însă tipul câmpului specificat (porneste de la 0!).

```
int mysql_field_len(resource rezultat, int  
                    offset_camp);
```

- ◆ Întoarce lungimea câmpului specificat (porneste de la 0!).

# Functii MySQL - Modificare

```
int mysql_affected_rows (resource  
    [id_conexiune] );
```

- ◆ Întoarce numărul de linii afectate de ultima cerere INSERT, UPDATE sau DELETE pe conexiunea specificată sau în lipsa acestui parametru pe ultima conexiune deschisă.

- ◆ Exemplu:

```
int mysql_affected_rows ($conexiune) ;  
int mysql_insert_id (resource  
    [id_conexiune] );
```

- ◆ Întoarce valoarea generată de ultimul INSERT executat pentru o coloană definită cu AUTO\_INCREMENT.

# Funcții MySQL - Terminare

```
int mysql_free_result(resource rezultat);
```

- ◆ Apelul acestei funcții eliberează spațiul ocupat de rezultatul unei cereri.

```
int mysql_close(resource [id_conexiune] );
```

- ◆ Închide conexiunea specificată sau în lipsă ultima conexiune deschisă.
- ◆ Întoarce *false* în caz de eșec, altfel *true*.
- ◆ Nu închide conexiunile deschise cu *mysql\_pconnect()*.
- ◆ Exemplu:

```
int mysql_close($conexiune);
```

# Funcții MySQL

- ◆ Pe lângă funcțiile prezentate există și altele care se pot consulta în documentația PHP.
- ◆ Unele dintre acestea vor fi prezentate în finalul capitolului, în contextul rezolvării problemelor de securitate (cum este `mysql_real_escape_string`)



# Funcții Oracle - conectare

```
resource oci_connect ( string user ,  
    string passw [, string bd [, string  
    setcar [, int mod_sesiune ]]) )
```

- ◆ Se realizează conectarea la serverul Oracle local cu userul și parola specificată.
- ◆ Dacă lipsește numele instanței (bd) PHP îl ia din variabila env. ORACLE\_SID.
- ◆ Setul de caractere se poate indica începând cu versiunea 9.2 de Oracle
- ◆ Mod\_sesiune poate fi **OCI\_DEFAULT**, **OCI\_SYSOPER** și **OCI\_SYSDBA**. Ultimele două stabilesc o sesiune privilegiată.
- ◆ Există și aliasul ocilogon(...) cu aceleași efecte.

# Funcții Oracle - conectare

```
resource oci_pconnect ( string user ,  
                        string passwd [, string bd [, string  
                        setcar [, int mod_sesiune]] ] )
```

- ◆ Are aceiași parametri ca precedenta.
- ◆ Funcția stabilește o conexiune persistentă, care nu se închide la terminarea scriptului și este reutilizată de scripturile următoare
- ◆ În felul acesta se micșorează overhead-ul de sistem.

# Funcții Oracle - conectare

```
resource oci_new_connect ( string user ,  
    string passw [, string bd [, string  
    setcar [, int mod_sesiune]] )
```

- ◆ Deschide o noua conexiune, chiar dacă există deja una (celelalte 2 funcții nu fac asta).
- ◆ Toate 3 returnează un descriptor de conexiune (echiv. TRUE) sau FALSE în caz de eroare

# Funcții Oracle - Parse

- ◆ Al doilea pas după conectare este compilarea (parsingul) cererii.
- ◆ Aceasta se face cu funcția:

```
resource oci_parse ( resource conexiune ,  
                    string cerere )
```

- ◆ Returnează un descriptor de cerere (statement handle) necesar execuției efective.
- ◆ Exemplu:

```
$c = oci_connect ("scott", "tiger");  
$cerere = "select * from STUD where  
         cods=" . $_REQUEST ['cods'] ;  
echo "<h2>Cererea este: $cerere</h2>";  
$stmt = oci_parse ($c, $cerere);
```

# Functii Oracle - Executie

- ◆ Executia efectiva a cererii se face cu:

```
bool oci_execute ( resource $stmt  
                  [, int mod])
```

- ◆ Parametrii sunt: un descriptor de cerere returnat de `oci_parse` si optional modul de tratare al tranzactiilor.
- ◆ Modul poate fi:  
**OCI\_COMMIT\_ON\_SUCCESS**  
(optiunea implicita) si **OCI\_DEFAULT**.

# Functii Oracle - Executie

## **OCI\_DEFAULT:**

- ◆ In al doilea caz, executia unei cereri DML duce la demararea unei tranzactii.
- ◆ Aceasta tranzactie va fi revocata automat la terminarea scriptului sau la inchiderea conexiunii.
- ◆ Pentru a controla aceasta tranzactie se poate folosi `oci_commit` si `oci_rollback`

# Functii Oracle - Executie

```
bool oci_commit ( resource conexiune);
```

```
bool oci_rollback ( resource  
                    conexiune);
```

- ◆ Efectul lor este de a comite, respectiv revoca tranzactia in curs pe acea conexiune.
- ◆ Aceasta tranzactie poate fi formata din executia uneia sau mai multor cereri DML.
- ◆ Returneaza TRUE / FALSE in caz de succes/esec.

# Functii Oracle - Rezultat

- ◆ Pentru a afla dimensiunea rezultatului se pot folosi:

```
int oci_num_rows ( resource $stmt )
```

- ◆ Intoarce numarul de linii afectate de executia cererii (cate linii returneaza un select dar si cate linii sterge un delete de exemplu)

```
int oci_num_fields ( resource $stmt )
```

- ◆ Returneaza numarul de coloane ale rezultatului executiei unei cereri SELECT.

```
string oci_field_name ( resource stmt , int  
coloana )
```

- ◆ Returneaza numele coloanei cu numarul specificat (prima coloana are indicele 1)



# Exemplu

```
. . . . .
$stmt = oci_parse($c, $cerere);
// executie
oci_execute($stmt, OCI_DEFAULT);
// rezultatul va fi prezentat ca tabela
echo "<table border = 1>";
// luam in nc nr. de coloane in rezultat
$nc = oci_num_fields($stmt);
// afisam antet tabel
echo "<tr>";
for($i=1;$i<=$nc;$i++)
    echo "<td>".oci_field_name($stmt, $i)."</td>";
echo "</tr>";
. . . . .
```

# Funcții Oracle - Fetch

- ◆ Pentru încărcarea liniilor există 4 funcții asemănătoare cu cele din MySQL:

```
array oci_fetch_row ( resource stmt )
```

- ◆ Intoarce tablou cu indici numerici

```
array oci_fetch_assoc ( resource stmt)
```

- ◆ Intoarce tablou asociativ

```
array oci_fetch_object ( resource stmt)
```

- ◆ Intoarce obiect

# Funcții Oracle - Fetch

```
array oci_fetch_array ( resource stmt [,  
                        int mod ] )
```

## ◆ Modul poate fi:

1. **OCI\_BOTH** - intoarce array asociativ si cu indici numerici - valoarea implicita.
2. **OCI\_ASSOC** - intoarce array asociativ
3. **OCI\_NUM** - intoarce array cu indici numerici
4. **OCI\_RETURN\_NULLS** - creaza elemente goale pentru valorile nule
5. **OCI\_RETURN\_LOBS** - returneaza valoarea LOB-ului unui descriptor

# Exemplu

```
<?php
    $connection = oci_connect("stud", "studpw");

    $cerere = "SELECT nume, pret, coloana_lob FROM
    produse";

    $stmt = oci_parse ($connection, $cerere);
    oci_execute ($stmt);

    while ($lin = oci_fetch_array ($stmt,
                                    OCI_RETURN_LOBS))
    {
        echo $lin[0]."<br>";
        echo $lin[1]."<br>";
        echo $lin['COLOANA_LOB']. "<br>";
        // ultima linie afiseaza continutul col. LOB
    }
?>
```

# Alt exemplu

- ◆ Continuarea exemplurilor anterioare:

```
while ($linie=oci_fetch_row($stmt)) {
    // incepen linie date
    echo "<tr>";
    for($i=0;$i<$nc;$i++)
    {
        // daca e valoare nula o inlocuim cu sirul NULL
        if (!isset($linie[$i]))
            $linie[$i] = "NULL";
        // afisam efectiv o celula
        echo "<td>".$linie[$i]."</td>";
    }
    // gata linie date
    echo "</tr>";
}
```

# PL/SQL

- ◆ Prezentam in continuare un exemplu de executie a unui bloc PL/SQL in PHP.
- ◆ Se foloseste functia `oci_bind_by_name` pentru punerea in corespondenta a variabilelor externe PL/SQL cu variabile PHP.
- ◆ Sintaxa functiei este (vezi si documentatia):

```
bool oci_bind_by_name ( resource stmt ,  
    string nume_oracle , mixed &var_PHP [,  
    int lung_maxima = -1 [, int tip =  
        SQLT_CHR ] ] )
```

# Fetch all

```
int oci_fetch_all ( resource stmt , array &output [,  
    int skip= 0 [, int maxlinii = -1 [, int $flags= 0  
                    ]]] )
```

- ◆ Incarca toate liniile unui rezultat SELECT (sau un numar specificat de linii) intr-un array.
- ◆ Intoarce numarul de linii incarcate
- ◆ Exemplu:

```
$conn = oci_connect(. . . );  
$stmt = oci_parse($conn, "select * from emp");  
oci_execute($stmt);  
$nrows = oci_fetch_all($stmt, $results);  
if ($nrows > 0) {  
    foreach ($results as $key => $val) {  
        echo "$key, $val\n";  
    }  
}
```

# PL/SQL

```
$c = oci_connect("stud", "studpw");
// variabile: $cati (PHP) si :cati (externa Oracle)
$cerere = "begin
    select count(*) into :cati from STUD
    where cods=".$_REQUEST['cods']."; "
    "exception
    when others then
        :cati := 0;
    end;"; // atentie: ; de dupa end!
$stmt = oci_parse($c, $cerere);
// bind by name:
oci_bind_by_name($stmt, ":cati", $cati, 32);
// executie
oci_execute($stmt, OCI_DEFAULT);
// rezultatul va fi in variabila PHP $cati
echo "Numar de studenti: ".$cati;
```



# Alt exemplu

```
$c = oci_connect("scott","tiger");
$cerere = "begin
    select cods, nume into :cods, :nume from STUD
    where matr=".$_REQUEST['matr']."; ".
    "exception
    when others then
        :cods := 0;
        :nume := 'nu avem astfel de student';
    end;"; // nu trebuie uitat ; dupa end
$stmt = oci_parse($c, $cerere);
// bind by name: se asociaza 2 variabile externe cu var. PHP
oci_bind_by_name($stmt, ":cods", $cods, 32);
oci_bind_by_name($stmt, ":nume", $nume, 32);
// executie
oci_execute($stmt,OCI_DEFAULT);
echo $cods, " ", $nume;
```

# Functii Oracle - Erori

- ◆ Descrierea erorilor se poate obtine cu:  
`array oci_error ([ resource $sursa ] )`
- ◆ Pentru erorile de conexiune nu se indica sursa
- ◆ Pentru celelalte se pune descriptorul cel mai apropiat
- ◆ Array-ul contine elementele:
  - ◆ code (codul de eroare)
  - ◆ message (mesajul)
  - ◆ offset (pozitia erorii)
  - ◆ sqltext (textul cererii eronate)

# Exemplu

```
if (!($r = oci_execute($stmt))) {  
    // se intra aici in caz de eroare  
    $e = oci_error($stmt);  
    echo htmlentities($e['message']);  
    echo "<pre>";  
    echo htmlentities($e['sqltext']);  
    printf("\n%".($e['offset']+1)."s", "  
    ^"); // o sageata la pozitia erorii  
    echo "</pre>";  
}
```

# Funcții Oracle - Final

- ◆ Inchiderea conexiunii se face cu:

```
bool oci_close ( resource conexiune )
```

- ◆ Pe lângă funcțiile prezentate mai există multe altele (vezi documentația)

# Elemente de securitate

- ◆ In continuare prezentam doar cateva elemente de securitate pentru aplicatii scrise in PHP si care acceseaza o baza de date.
- ◆ Una dintre problemele principale este aceea ca unui script care in mod obisnuit este asociat unui formular i se pot transmite parametri - inclusiv in bara de adresa (metoda GET) - care nevalidati pot duce la injectie SQL.

# Injectia SQL

- ◆ Injectia SQL inseamna executia unei cereri nedorite inserata intr-un sir de caractere executat de serverul de baze de date.
- ◆ Ea apare ca urmare a nevalidarii datelor de intrare si a ignorarii unor masuri minime de securitate

# Comentarii

- ◆ Semnul de comentarii poate afecta cererea SQL executata cu un parametru primit.
- ◆ Exemplu: fie o cerere de tipul  

```
SELECT * FROM members WHERE username =  
'param1' AND password = 'param2'
```
- ◆ Daca pentru param1 se introduce valoarea: admin'-- cererea executata va fi:

# Comentarii

```
SELECT * FROM members WHERE username =  
    'admin' -- AND password = 'param2'
```

- ◆ Se observa ca partea de verificare parola din cerere este acum comentata - deci nu se mai verifica si parola ci doar username.



# Comentarii

- ◆ Variante (SqlServer, MySQL, Oracle):
  - ◆ admin' --
  - ◆ admin' #
  - ◆ admin'/\*
  - ◆ ' or 1=1--
  - ◆ ' or 1=1#
  - ◆ ' or 1=1/\*
  - ◆ ') or '1'='1--
  - ◆ ') or ('1'='1--

# Date fara '

- ◆ O recomandare general acceptata este aceea de a pune intre apostrofi toate elementele primite ca intrare de la user.
- ◆ Iata un exemplu (bine si mai putin bine)

```
$query = "SELECT * FROM tabela where col  
= ' " . $_REQUEST['ceva'] . "'";  
$query = "SELECT * FROM tabela where col  
= " . $_REQUEST['ceva'] ;
```

# Date fara '

- ◆ Ce se poate intampla in al doilea caz:

- ◆ Input: 1 or (1=1). Cererea devine:

```
$query = "SELECT * FROM tabela where col  
= 1 or (1=1)"
```

- ◆ Vor fi intoarse deci toate liniile tabeli si nu doar cateva.

- ◆ Input: 1 and (1=0); drop table tabela;

- ◆ Cererea devine:

```
$query = "SELECT * FROM tabela where col  
= 1 and (1=0); drop table tabela; "
```

# Cereri multiple

- ◆ In acest caz sirul de executat contine doua cereri SQL, dintre care a 2-a este distructiva.
- ◆ Exista sisteme de gestiune care executa astfel de succesiuni de cereri primite din PHP (postgresql, mssql)
- ◆ Solutia este fie folosirea apostrofilor fie folosirea mecanismului de marcare a unor caractere speciale (inclusiv apostrof).

# Escape string

- ◆ Daca parametrul PHP (din PHP.ini) `magic_quotes_gpc` este trecut pe `On`, toate aceste caractere speciale sunt prefixate cu backslash.
- ◆ Dupa inlaturarea acestuia, se poate folosi mecanismul propriu de escape al SGBD-ului respectiv.
- ◆ Iata un exemplu:

# Exemplu

```
<?php
    if (get_magic_quotes_gpc()) {
        $nume = stripslashes($nume);
        $descriere = stripslashes($descriere);
// Cererea:
    query = sprintf(
        "SELECT * FROM users WHERE user='%s'
        AND password='%s'",
        mysql_real_escape_string($nume),
        mysql_real_escape_string($descriere));
. . . . .
    ?>
```

# Exemplu

- ◆ Functia `mysql_real_escape_string` va prefixa cu `\` caracterele speciale din MySQL: `\x00`, `\n`, `\r`, `\,`, `'`, `"` si `\x1a`.
- ◆ Analog, prin folosirea primei sintaxe (cu `'`), cele doua cereri anterioare devin:

```
$query = "SELECT * FROM tabela  
where col = '1 or (1=1)'"
```

```
$query = "SELECT * FROM tabela  
where col = '1 and (1=0); drop  
table tabela;'"
```

# Bibliografie

- ◆ Documentatia PHP

<http://www.php.net/docs.php>

## Cateva carti disponibile online:

- ◆ Chris Newman: Sams - Teach Yourself PHP in 10 Minutes(2005)

[http://www.net130.com/CMS/Pub/book/book\\_web/book\\_web\\_php/2005\\_10\\_19\\_70383.htm](http://www.net130.com/CMS/Pub/book/book_web/book_web_php/2005_10_19_70383.htm)

- ◆ Ilia Alshanetsky: PHP Architects Guide to PHP Security (si altele), la adresa:

<http://cid-846ffdcf0d3320d8.skydrive.live.com/browse.aspx/eBook>

- ◆ SQL Injection Cheat Sheet

<http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>