

# Aplicații cu BD

Call-Level Interface

Java Database Connectivity

PHP

# Intercalare de SQL

- Interogările SQL sunt adeseori construite de către programe.
- Aceste interogări primesc *constante* introduse de utilizator.
- Atunci când codul nu este tratat cu atenție, se pot întâmpla lucruri neașteptate.

# Exemplu: Intercalare SQL

- Presupunem că există relația

Utilizatori(**nume**, **parolă**, **cont**)

- Se construiește o interfață Web : se citește numele și parola utilizatorului, în șirurile *n* și *p*, se emite interogarea, se afișează numărul contului.

```
SELECT cont FROM Utilizatori  
WHERE nume = :n AND parola = :p
```

# Ecranul utilizator:

Nume:

Misu'

--

Comentariu  
în Oracle sau  
MS SQL Server

Parola:

interesant?

Numărul contului este 1234-567

# Interogarea la Execuție

```
SELECT cont FROM Utilizatori
```

```
WHERE nume = 'Misu' --' AND
```

```
parola = 'interesant?'
```

Sunt tratate ca și comentariu



# Limbaaj Gazdă/ Interfețe SQL Via Biblioteci

- A treia abordare pentru conectarea la BD a limbajelor convenționale este apelul prin intermediul bibliotecilor.

1. C + CLI
2. Java + JDBC
3. PHP + PEAR/DB

# Arhitectura "Three-Tier"

- Un mediu obișnuit de utilizare a BD are trei straturi de procesoare:
  1. *Servere Web* --- tratează cu utilizatorul.
  2. *Servere de Aplicație* --- execută "business logic".
  3. *Servere BD* --- extrag din BD ceea ce este necesar pentru serverele de aplicație.

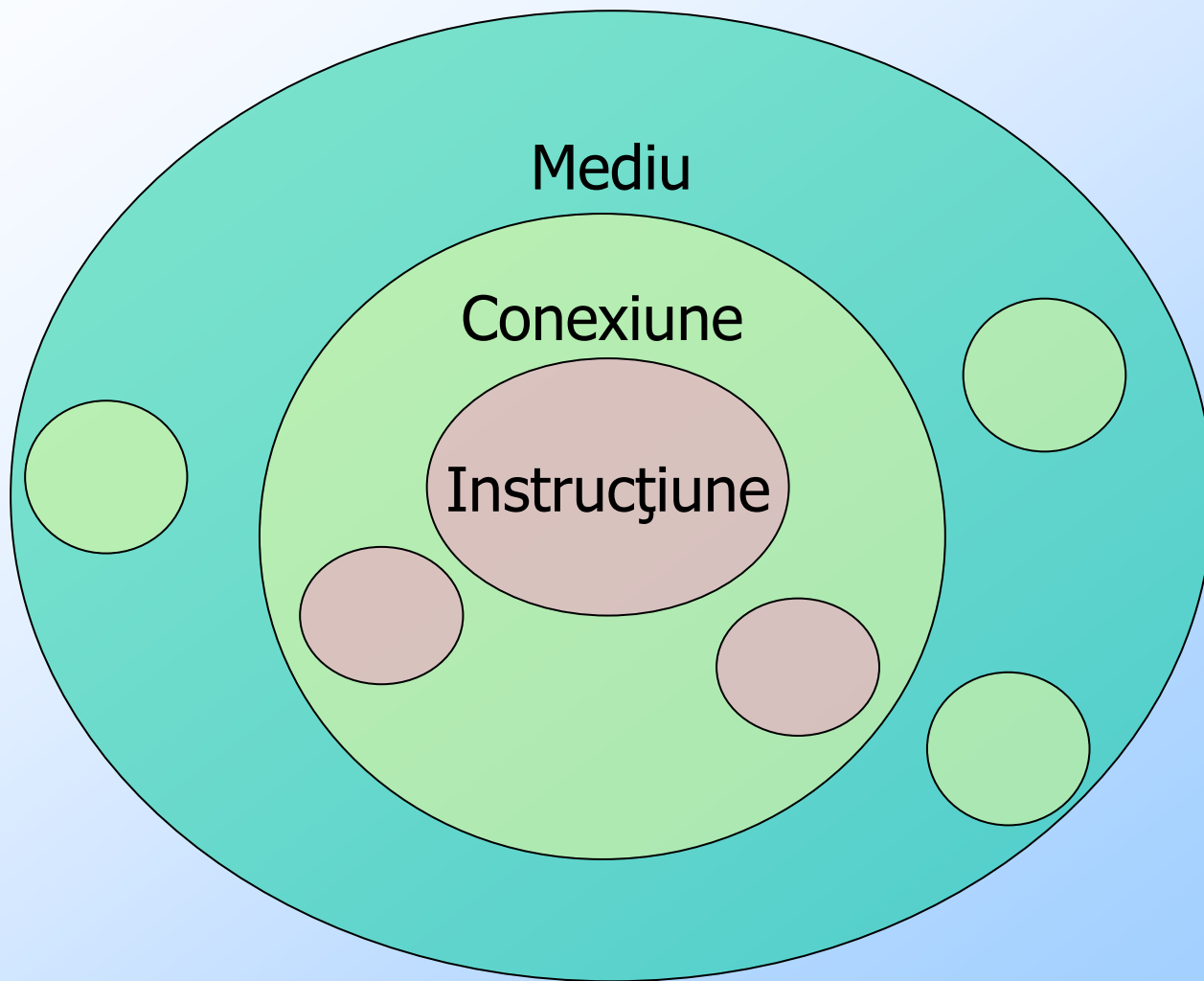
# Exemplu: Amazon

- BD păstrează informația despre produse, clienți, etc.
- “Business logic” include printre altele “ce trebuie făcut după ce este selectat ‘checkout’?”
  - Răspuns: Să se afișeze ecranul “cum veți achita?”.



# Medii, Conexiuni, Interogări

- BD este, pentru majoritatea limbajelor de acces la baze de date, un *mediu* ("*environment*").
- Serverele BD întrețin un număr de *conexiuni*, astfel încât serverele de aplicații pot lansa interogări sau pot efectua actualizări.
- Serverul de aplicație emite de obicei *instrucțiuni* : interogări și actualizări.



# SQL/CLI

- În loc să se folosească un preprocesor (embedded SQL), se poate folosi o bibliotecă cu funcții.
  - Biblioteca pentru C este numită SQL/CLI = "*Call-Level Interface*."
  - Preprocesorul pentru embedded SQL translatează instrucțiunile EXEC SQL ... în CLI sau apeluri similare.

# Structuri de Date

□ C se conectează la BD cu structuri de tipul următor:

1. *Medii* : reprezintă SGBD-uri instalate.
2. *Conexiuni* : conexiuni (login) la BD.
3. *Instrucțiuni* : instrucțiuni SQL transmise printr-o conexiune.
4. *Descrieri* : înregistrări despre tuplele unei interogări, sau parametrii unei instrucțiuni.

# Handle-uri

- Funcția **SQLAllocHandle(T,I,O)** este folosită pentru a crea aceste structuri, ce se numesc *handle-uri* de mediu, conexiune și instrucțiune.
  - $T$  = tip, de exemplu, SQL\_HANDLE\_STMT.
  - $I$  = handle de intrare = structură la un nivel superior (instrucțiune < conexiune < mediu).
  - $O$  = (adresa unui) handle de ieșire.

# Exemplu: SQLAllocHandle

```
SQLAllocHandle(SQL_HANDLE_STMT,  
               con_Mea, &inst_Mea);
```


- Con\_Mea este un handle de conexiune creat anterior.
- inst\_Mea este numele unui handle de instrucțiune ce va fi creat.

# Pregătirea și Executarea

- **SQLPrepare( $H, S, L$ )** cauzează interpretarea șirului  $S$ , de lungime  $L$ , ca fiind o instrucțiune SQL și optimizarea ei;
  - instrucțiunea executabilă este plasată în handle-ul de instrucțiune  $H$ .
- **SQLExecute( $H$ )** cauzează execuția instrucțiunii SQL reprezentată de handle-ul de instrucțiune  $H$ .

# Exemplu: Pregătire și Execuție

```
SQLPrepare(inst_Mea, "SELECT  
beer, price FROM Sells  
WHERE bar = 'Joe''s Bar'",  
SQL_NTS);  
SQLExecute(inst_Mea);
```



Această constantă precizează că argumentul al doilea este "null-terminated string"; adică, lungimea este determinată prin numărarea caracterelor.



# Execuția Directă

- Dacă o instrucțiune  $S$  va fi executată doar o singură dată, se poate combina PREPARE și EXECUTE cu:

**SQLExecuteDirect( $H, S, L$ );**

- Unde,  $H$  este un handle de instrucțiune și  $L$  este lungimea șirului  $S$ .

# Extragerea Tuplelor

- La execuția instrucțiunii SQL, dacă aceasta este o interogare, este nevoie să se extragă tuplele rezultat.
  - Un cursor este implicit prin faptul că s-a executat o interogare, cursorul nu trebuie declarat explicit.
- **SQLFetch(H)** obține următoarea tuplă din rezultatul instrucțiunii cu handle-ul *H*.

# Accesarea Rezultatului Interogării

- La extragerea unei tuple, este nevoie să se preia componentele tuplei undeva.
- Fiecare componentă este legată la o variabilă prin funcția **SQLBindCol**.
- Această funcție are 6 argumente, din care vor fi prezentate doar componentele 1, 2 și 4:
  - 1 = handle-ul instrucțiunii interogare.
  - 2 = numărul coloanei.
  - 4 = adresa variabilei.

# Exemplu: "Binding"

- Presupunem că s-a executat apelul `SQLExecute(inst_Mea)`, unde `inst_Mea` este handle-ul interogării:

```
SELECT beer, price FROM Sells  
WHERE bar = 'Joe' 's Bar'
```

- Legarea rezultatului la Berea și Pretul:  
`SQLBindCol(inst_Mea , 1, , &Berea, , );`  
`SQLBindCol(inst_Mea , 2, , &Pretul, , );`

# Exemplu: "Fetching"

- În acest moment se pot extrage toate tuplele răspunsului:

```
while ( SQLFetch(inst_Mea ) != SQL_NO_DATA)
{
    /* aici se utilizează Berea și Pretul */
}
```

Macrou CLI ce reprezintă  
SQLSTATE = 02000 = "failed  
to find a tuple."

# JDBC

- *Java Database Connectivity* (JDBC) este o bibliotecă similară cu SQL/CLI, dar cu Java ca limbaj gazdă.
- Asemănător CLI, dar cu puține diferențe.

# Specificarea unei Conexiuni

```
import java.sql.*;  
Class.forName("com.mysql.jdbc.Driver");  
Connection con_Mea =  
    DriverManager.getConnection(...);
```

The diagram illustrates the components of a JDBC connection setup. It features four colored boxes highlighting specific parts of the code: a teal box for the import statement, a pink box for the Class.forName call, a blue box for the DriverManager class, and a yellow box for the getConnection method's parameters. Arrows point from these boxes to descriptive labels: 'Clasele JDBC' points to the import statement; 'Cel încărcat cu forName' points to the DriverManager box; 'URL-ul BD, numele și parola' points to the getConnection parameters box; and 'Driver pentru mySql' points to the Class.forName call.

Clasele JDBC

Cel încărcat  
cu forName

URL-ul BD,  
numele și  
parola

Driver  
pentru  
mySql

# Instrucțiuni

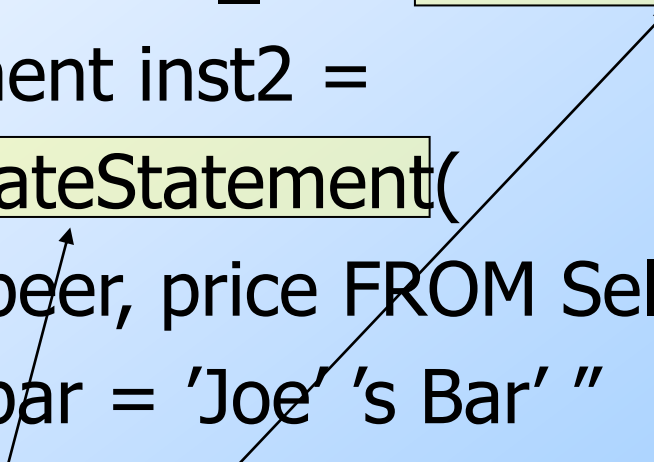
- JDBC oferă două clase:
  1. *Statement* = un obiect ce poate accepta un string ce este o instrucțiune SQL și poate executa un astfel de string.
  2. *PreparedStatement* = un obiect ce are o instrucțiune SQL asociată, ce este gata de execuție.



# Crearea Instrucțiunilor

- Clasa "Connection" are metode pentru a crea "Statements" și "PreparedStatement".

```
Statement inst1 = con_Mea.createStatement();  
PreparedStatement inst2 =  
    con_Mea.createStatement(  
        "SELECT beer, price FROM Sells " +  
        "WHERE bar = 'Joe' 's Bar' "  
    );
```



**createStatement** fără argumente returnează un obiect "Statement"; iar cu un argument returnează un obiect "PreparedStatement".

# Execuția instrucțiunilor SQL

- JDBC face distincție între interogări și actualizări, pe care le numește “updates.”
- “Statement” și “PreparedStatement” au fiecare metodele `executeQuery` și `executeUpdate`.
  - Pentru “Statements”: un argument - interogarea sau actualizarea ce trebuie executată.
  - Pentru “PreparedStatement”: nici un argument.

# Exemplu: Actualizare

- inst1 este un obiect "Statement".
- El se poate utiliza pentru a adăuga o tuplă:  

```
inst1.executeUpdate(  
    "INSERT INTO Sells " +  
    "VALUES ('Brass Rail', 'Bud', 3.00) "  
);
```

# Exemplu: Interogare

□ inst2 este un obiect "PreparedStatement" ce păstrează interogarea:

"SELECT beer, price FROM Sells WHERE bar = 'Joe's Bar' "

□ **executeQuery** returnează un obiect din clasa ResultSet, ce va fi examinat ulterior.

□ Interogarea:

ResultSet meniu = inst2.executeQuery();

# Accesul la "ResultSet"

- Un obiect de tipul ResultSet este ceva asemănător unui cursor.
- Metoda `next()` avansează "cursorul" la următoarea tuplă.
  - Prima dată când se aplică `next()`, se regăsește prima tuplă.
  - Dacă nu mai există tuple, `next()` returnează valoarea `false`.

# Accesul la Componentele Tuplelor

- Atunci când "ResultSet" face referire la o tuplă, se pot obține componentele tuplei aplicând anumite metode pentru "ResultSet".
- Metoda `getX(i)`, unde  $X$  este un anumit tip, și  $i$  este numărul componentei, returnează valoarea componentei.
  - Valoarea trebuie să aibă tipul  $X$ .

# Exemplu: Accesarea Componentelor

□ Meniu = "ResultSet" pentru interogarea

"SELECT beer, price FROM Sells WHERE bar = 'Joe' 's Bar' "

□ Sunt accesate "beer" și "price" pentru fiecare tuplă:

```
while ( meniu.next() ) {  
    Berea = Menu.getString(1);  
    Pretul = Menu.getFloat(2);  
    /*anumite operatii cu Berea și Pretul*/  
}
```

# PHP

- Un limbaj ce este folosit pentru acțiuni într-un text HTML.
- Este indicat de `<? PHP cod?>`.
- Există biblioteca DB în *PEAR* (PHP Extension and Application Repository).
  - Este inclusă cu `include (DB.php)`.



# Variabile în PHP

- Încep cu \$.
- Este permis să nu se declare tipul unei variabile.
- Se atribuie unei variabile o valoare ce aparține unei "clase", caz în care sunt accesibile metode ale acelei clase.

# Valori "String"

- PHP rezolvă o problemă foarte importantă pentru limbajele ce construiesc şirurile de caractere ca şi valori:
  - Cum se spune că un subşir trebuie interpretat ca variabilă şi să fie înlocuit (subşirul) cu valoarea sa?
- Soluţia PHP:
  - Ghilimele semnifică "se înlocuieşte";
  - Apostrofuri semnifică "nu se înlocuieşte".

## Exemplu: Se înlocuiește sau Nu?

```
$100 = "una sută dolari";
```

```
$sue = 'îmi datorezi $100.';
```

```
$joe = "îmi datorezi $100.";
```

□ Valoarea variabilei **\$sue** este

    'îmi datorezi \$100',

□ În timp ce valoarea variabilei **\$joe** este

    'îmi datorezi una sută dolari'.

# Tablouri PHP

- Există două categorii: *numerice* și *asociative*.
- Tablourile numerice sunt cele obișnuite, indexate 0,1,...
  - Exemplu: `$a = array("Paul", "George", "John", "Ringo");`
    - Atunci `$a[0]` este "Paul", `$a[1]` este "George", ș.a.m.d.

# Tablouri Asociative

- Elementele unui tablou asociativ \$a\$ sunt perechi  $x \Rightarrow y$ , unde  $x$  este un șir de caractere cheie și  $y$  este orice valoare.
- Dacă  $x \Rightarrow y$  este un element al lui \$a\$, atunci \$a[x]\$ este  $y$ .

# Exemplu: Tablouri Asociative

- Un mediu poate fi exprimat ca un tablou asociativ:

```
$med_Meu = array(  
    "phptype"    => "mysql",  
    "hostspec"   => "localhost",  
    "database"   => "scoala",  
    "username"   => "student",  
    "password"   => "nuSeStie");
```

# Efectuarea unei Conexiuni

- Se importă biblioteca DB și se folosește tabloul \$med\_Meu:

```
$con_Mea = DB::connect($med_Meu);
```

Funcția connect  
din biblioteca DB

Clasa variabilei este Connection  
din cauză că este returnată  
de DB::connect().

# Execuția instrucțiunilor SQL

- Se aplică metoda **query** unui obiect Connection.
- Această metodă primește un argument de tipul string și returnează un rezultat.
  - Ce poate fi un cod de eroare sau relația returnată de o interogare.



# Exemplu: Execuția unei Interogări

- Să se găsească barurile ce vând o bere dată de variabila \$bere.

```
$bere = 'Bud';
```

```
$rezultat = $con_Mea->query(  
    "SELECT bar FROM Sells"  
    "WHERE beer = $bere ;") ;
```

Concatenare  
în PHP

De reținut că  
variabila se înlocuiește  
cu valoarea sa.

Aplicarea  
metodei

# Cursor în PHP

- Rezultatul unei interogări *este* reprezentat de tuplele returnate.
- Metoda **fetchRow** se aplică rezultatului și returnează următoarea tuplă, sau FALSE dacă nu mai există tuple.

# Exemplu: Cursor

```
while ($bar =  
    $result->fetchRow()) {  
    // diferite acțiuni cu $bar  
}
```

# Exemplu: testora.php

```
<?php
$c = oci_connect('student1', 'student', '//serverora/orcl');
$s = oci_parse($c, 'select city from locations');
oci_execute($s);
while ($res = oci_fetch_array($s, OCI_ASSOC))
{
    echo $res['CITY'] . "<br>";
}
?>
```

# Variabile BIND

- Aplicația re-execută instrucțiuni cu valori diferite
- Se îmbunătățește database throughput
- Ajută la prevenirea atacurilor de tip “SQL Injection”

# Variabile BIND

```
$s = oci_parse($c, "select last_name from employees  
                    where employee_id = :eidbv");  
$myeid = 101;  
oci_bind_by_name($s, ":EIDBV", $myeid);  
oci_execute($s);  
oci_fetch_all($s, $res);  
echo "Numele: ". $res['LAST_NAME'][0] . "<br>\n";  
$myeid = 102;  
oci_execute($s); // Nu necesită "re-parse"  
oci_fetch_all($s, $res);  
echo "Last name is: ". $res['LAST_NAME'][0] . "<br>\n";
```

# PL/SQL și PHP

create or replace procedure

myproc(a in varchar2, b in number) as

begin

insert into mytable (mydata, myid) values (a, b);

end;

/

<?php

\$c = oci\_connect('student1', 'student', '//serverora/orcl');

\$s = oci\_parse(\$c, "call myproc('mydata', 123)");

oci\_execute(\$s);

?>

# Exemplu: Tranzacție

```
function do_transactional_insert($c, $a)
{
    $s = oci_parse($c, 'insert into ptab (pdata) values
        (:bv)');
    oci_bind_by_name($s, ':bv', $v, 20, SQLT_CHR);
    foreach ($a as $v)
    $r = oci_execute($s, OCI_DEFAULT); // Nefinalizat
    oci_commit($c); // Finalizat
}
```