Web-technológia II.

Adatbázis-kezelés: DBA, MySQLi, PDO, ORM

Hatwágner F. Miklós

Széchenyi István Egyetem, Győr

2019. április 11.

DBA: DataBase Abstraction layer

- fájlokban tárolt kulcs-érték párok (Berkeley DB stílus)
- absztrakciós réteg → kapcsolódás többféle AB-kezelőhöz

Kezelő	Megjegyzések
gdbm	GNU database manager
db4	Oracle Berkeley DB 4 vagy 5 rendszerekhez
cdb	konstans adatbázisok
inifiles	A php.ini-hez hasonló fájlokhoz
qdbm	FAL Labs

Milyen kezelők használhatóak?

- dba_handlers()
- phpinfo()

Adatfeldolgozás lépései:

1. kapcsolódás: dba_open(), dba_popen()
2. műveletek elvégzése, pl.
   a. új kulcs-érték pár beszúrása: dba_insert()
   b. beszúrás / módosítás: dba_replace()
   c. törlés: dba_delete()
   d. tárolt párok bejárása:
      i. első kulcs lekérése: dba_firstkey()
      ii. további kulcsok lekérése: dba_nextkey()
      iii. kulcshoz tartozó érték lekérése: dba_fetch()
   e. kulcs létezésének ellenőrzése: dba_exists()
3. kapcsolat lezárása: dba_close()

Néhány további hasznos függvény:

- törlésre megjelölt rekordok tényleges eltávolítása: dba_optimize()
- memória $\leftrightarrow$ háttértár szinkronizáció: dba_sync()

Néhány megnyitási (`dba_open()`) üzemmód:

r olvasás

w olvasás/írás, létező fájlon

c olvasás/írás, létrehozza a fájlt, ha nem létezik

n létrehozás/csonkolás, majd olvasás/írás

A visszatérési értékkel lehet később az AB-ra hivatkozni.

Megjegyzések a mintapéldához:

- a szerializáció módjának testreszabása:
  - __sleep() visszaadhatja a szerializálandó adattagok tömbjét, de az öröklött privát adattagokra nem lehet hivatkozni
  - Serializable interfész megvalósítása
- a serialize() akár adattagok tömbjével is boldogul
- az unserialize() által visszaadott tömbből pl. a list() (nyelvi elem, PHP 5 és 7 esetén eltérően működik!) készíthet újra adattagokat
- dátum és idő kezelése:
  - DateTime osztály, formázás format() metódussal, a formátumsztring elemei.
  - Nyelvfüggő módon az strftime függvénnyel (ld. még setlocale)

dba/Hallgato.php

```php
<?php
  declare(strict_types = 1);

  class Hallgato implements \Serializable {
    private $nev;
    private $neptun;
    private $szuldatum;

    public function __construct(string $n, string $k, DateTime $d) {
      $this->nev = $n;
      $this->neptun = $k;
      $this->szuldatum = $d;
    }

    public function getNev() : string {
      return $this->nev;
    }
```

dba/Hallgato.php

```
public function getNeptun() : string {
  return $this->neptun;
}

public function setNeptun(string $neptun) {
  $this->neptun = $neptun;
}

public function getSzuldatum() : DateTime {
  return $this->szuldatum;
}

public function __toString() :string {
  return "Hallgató neve: {$this->nev}, ".
         "neptun kódja: {$this->neptun}, ".
         "születési dátuma: ".
         strftime("%Y. %B %e.", $this->szuldatum->getTimestamp());
}
```

dba/Hallgato.php

```
    public function serialize() : string {
      return serialize([
        $this->nev,
        $this->szuldatum
      ]);
    }

    public function unserialize($adat) {
      list(
        $this->nev,
        $this->szuldatum
      ) = unserialize($adat);
    }
  }
?>
```

dba/Evfolyam.php

```
<?php
  declare(strict_types = 1);

  class Evfolyam {
    private $db;

    public function __construct() {
      $this->db = new Adatbazis();
    }

    public function urlap() {
      echo <<<HTML
    <form method="post" action="{$_SERVER["PHP_SELF"]}">
      <fieldset><legend>Válasszon üzemmódot!</legend>
      <input type="radio" id="beszuras" name="mod" value="beszur" />
        <label for="beszuras">Beszúrás</label>
      <input type="radio" id="modositas" name="mod" value="csere"
        checked="checked" /><label for="modositas">Módosítás</label>
      <input type="radio" id="torles" name="mod" value="torles" />
        <label for="torles">Törlés</label>
      </fieldset>
```

dba/Evfolyam.php

```
      <fieldset><legend>Adja meg a hallgató adatait!</legend>
      <label for="neptun">Neptun kód:</label>
        <input type="text" id="neptun" name="neptun" /><br />
      <label for="nev">Név:</label>
        <input type="text" id="nev" name="nev" /><br />
      <label for="szuldatum">Születési dátum:</label>
        <input type="text" id="szuldatum" name="szuldatum" /><br />
      </fieldset>
      <p><input type="submit" value="OK" /></p>
    </form>

HTML;
    }

    public function feldolgozas() {
      if(isset($_POST["mod"]) &&
        in_array($_POST["mod"], ['beszur', 'csere', 'torles'])) {
        $this->db->{$_POST["mod"]}(new Hallgato(
          $_POST["nev"], $_POST["neptun"],
          new DateTime($_POST["szuldatum"]))); }
    }
```

dba/Evfolyam.php

```
    public function lista() {
      echo "<h1>A hallgatók listája:</h1>\n";
      echo $this->db;
    }
  }
?>
```

dba/Adatbazis.php

```
<?php
  declare(strict_types = 1);

  class Adatbazis {
    private $dbh = false;
    // Figyeljünk arra, hogy legyen írási jogunk!
    const DBNEV = "/home/feltoltes/www/hlista";

    public function __construct() {
      $this->dbh = dba_open(self::DBNEV, "c", "db4") or
        die("Nem sikerült megnyitni az adatbázist.");
    }
```

dba/Adatbazis.php

```
public function __destruct() {
  if($this->dbh != false) {
    dba_close($this->dbh);
  }
}

public function __toString() : string {
  $s = "";
  $kulcs = dba_firstkey($this->dbh);
  while($kulcs != false) {
    $obj = unserialize(dba_fetch($kulcs, $this->dbh));
    $obj->setNeptun($kulcs);
    $s .= "<p>$obj</p>\n";
    $kulcs = dba_nextkey($this->dbh);
  }
  return $s;
}
```

dba/Adatbazis.php

```
    public function beszur(Hallgato $h) {
      dba_insert($h->getNeptun(), serialize($h), $this->dbh) or
      print("<p>Nem sikerült a beszúrás.</p>\n");
      // Az echo-nak nincs visszatérési értéke!
    }

    public function csere(Hallgato $h) {
      dba_replace($h->getNeptun(), serialize($h), $this->dbh) or
      print("<p>Nem sikerült az objektum módosítása.</p>\n");
    }

    public function torles(Hallgato $h) {
      if(dba_exists($h->getNeptun(), $this->dbh)) {
      dba_delete($h->getNeptun(), $this->dbh) or
        print("<p>Nem sikerült az objektum törlése.</p>\n");
      } else {
        echo "<p>A hallgató nem szerepel az adatbázisban.</p>\n";
      }
    }
  }
?>
```

dba/index.php

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Database Abstraction Layer</title>
  </head>
  <body>
    <?php
      setlocale(LC_ALL, "hu_HU.UTF-8");

      spl_autoload_register(function ($osztaly) {
        include_once($osztaly.".php");
      });

      $ef = new Evfolyam();
      $ef->feldolgozas();
      $ef->lista();
      $ef->urlap();
    ?>
  </body>
</html>
```

MySQLi

MySQLi: natív API MySQL 4.1 és újabb relációs adatbázis-kezelőkhöz

- Régebbi MySQL verziók: MySQL API, elavult
- MySQLi = MySQL Improved, néhány előnye a régi API-val szemben:
  - OO szemléletben is használható
  - előkészített kifejezések használata (prepared statement)
  - tranzakciókezelés
  - kibővített nyomkövetési lehetőségek
- AB-kezelés lépései:
  1. kapcsolódás a szerverhez
  2. SQL utasítások küldése
  3. kapcsolat lezárása

Perzisztens kapcsolat

- AB kapcsolat felépítése erőforrás-igényes, mértéke függhet:
  - adatbázis típusától,
  - a webszervertől független hardveren fut-e,
  - mekkora terheléssel futnak a szerverek, stb.
- Extra terhelés elkerülése: állandó (perzisztens) kapcsolattal
  - „ugyanolyan” kapcsolathoz egy korábban már felépítettet próbál újrahasznosítani
  - „ugyanolyan” = azonos szerver, felhasználó, jelszó
  - funkcionális előnye nincs, de a teljesítményt fokozhatja
- Mikor valósítható meg az állandó kapcsolat? → PHP futtatás módjától függ
  1. CGI burkoló segítségével: minden oldalletöltéskor újra kell indítani az értelmezőt → állandó kapcsolat nem valósítható meg, de nem okoz hibát, ha a szkript ilyet kér
  2. Apache modul → OK
  3. Többszálú webszerver bővítményeként, pl. Microsoft IIS → OK

Perzisztens kapcsolat

- Hátrányok, lehetséges problémák:
  - az AB szerveren korlátozható a kapcsolatok száma (pool) $\rightarrow$ ezt elérve a kapcsolódás sikertelen
  - szoftverhibák (pl. végtelen) ciklus $\rightarrow$ a kapcsolat soha nem fog mások rendelkezésére állni
  - teljes tábla zárolás (pl. MyISAM) után a feloldás elmaradása $\rightarrow$ többi kapcsolat blokkolva várakozik
  - hasonló problémát okoz, ha a szkript hamarabb áll le (programhiba), mint a tranzakció, amit megkezdett

Állandó kapcsolat könnyen megvalósítható, pl. MySQLi-nél a hosztnév elé p:-ot kell tenni

Kapcsolódás a szerverhez

```
mysqli::__construct (
  [ string $host = ini_get("mysqli.default_host")
  [, string $username = ini_get("mysqli.default_user")
  [, string $passwd = ini_get("mysqli.default_pw")
  [, string $dbname = ""
  [, int $port = ini_get("mysqli.default_port")
  [, string $socket = ini_get("mysqli.default_socket")
  ]]]]]] )
```

ini_get() kulcs-érték pár olvasása a php.ini fájlból

host a szerver neve, vagy IP-cím

username felhasználónév az AB szerveren

passwd jelszó ugyanott

dbname adatbázis (séma) amihez csatlakozni szeretnénk

Kapcsolódási hibák kezelése

- @ operátorral elnyomható egy tetszőleges kifejezés által generált hibák kijelzése
- `mysqli::$connect_errno` kapcsolódási hiba kódja
- `mysqli::$connect_error` szöveges kapcsolódási hibaüzenet

Bármilyen SQL utasítás kiadható:

mysqli::query (string $query [, int $resultmode = MYSQLI_STORE_RESULT])

MYSQLI_STORE_RESULT eredménytábla pufferezett tárolása

MYSQLI_USE_RESULT eredménytáblát nem pufferezi → nagy táblák kezelésénél hasznos, de

1. csak egy irányban, egyszer járhatóak be az eredménytábla sorai
2. az eredménytábla lezárása (close()) előtt más utasítás nem küldhető a szervernek

Eredménytábla: mysqli_result példányban

- mysqli_result::fetch_object() az eredménytábla aktuális sorának adataiból hozza létre a paraméterként adott típusú objektumot (ennek hiányában: stdClass). Az oszlopnévvel egyező nevű adattagba teszi az adatot, majd a belső sormutatót lépteti. NULL-lal tér vissza, ha nincs több sor. Az adattagok hamarabb kapnak értéket, mint ahogy az alapértelmezett konstruktort hívja.
- mysqli_result::fetch_row() számokkal indexelt tömbként adja vissza a sor adatait
- mysqli_result::fetch_assoc() mezőnevek a kulcsok
- mysqli_result::fetch_array() mindkét kulcsolást létrehozza
- Traversable interfészt megvalósítja → foreach
- mysqli_result::data_seek() ugrik a tábla adott sorára (csak pufferezett esetben)
- mysqli_result::$num_rows eredmény sorok száma (SELECT)
- mysqli_result::free() felszabadítja az eredménytáblát

További hasznos metódusok, adattagok:

- mysqli::$affected_rows, mysqli_stmt::$affected_rows érintett sorok száma (INSERT/UPDATE/DELETE)
- mysqli::$insert_id, mysqli_stmt::$insert_id utoljára generált ID értéke
- mysqli::close() kapcsolat lezárása, PHP szkript lefutásával egyébként is megtörténik

MySQLi

Előkészített kifejezések, pl. sok rekord gyors felviteléhez

1. SQL sablon előkészítése, sablon küldése
2. sablonba helyettesítendő változók értékének küldése

Sablon előkészítése: mysqli_stmt mysqli::prepare()

- mysqli_stmt::bind_param() változók ?-lel jelölt helyre történő bekötése a sablonba (pl. tábla nevét nem lehet helyettesíteni); paraméterek: típusoknak megfelelő string (string, integer, double, blob), helyettesítendő változók
- mysqli_stmt::execute() kérés küldése
- mysqli_stmt::$affected_rows utolsó lekérdezés által érintett sorok száma (INSERT/UPDATE/DELETE)
- mysqli_stmt::bind_result() eredménytábla (SELECT) oszlopainak változókhoz kötése
- mysqli_stmt::fetch() bekötött változókba helyezi a következő sor adatait

mysqli/Hallgato.php

```
<?php
  declare(strict_types = 1);

  class Hallgato {
    private $nev;
    private $neptun;
    private $szuldatum;

    public function __construct(string $n=null, string $k=null,
                                DateTime $d=null) {
      if($n !== null) {
        $this->nev = $n;
        $this->neptun = $k;
        $this->szuldatum = $d;
      } else {
        $this->szuldatum = new DateTime($this->szuldatum);
      }
    }
```

mysqli/Hallgato.php

```
    public function getNev() : string {
      return $this->nev;
    }

    public function getNeptun() :string {
      return $this->neptun;
    }

    public function getSzuldatum() : string {
      return $this->szuldatum;
    }

    public function __toString() : DateTime {
      public function __toString() :string {
        return "Hallgató neve: {$this->nev}, ".
             "neptun kódja: {$this->neptun}, ".
             "születési dátuma: ".
             strftime("%Y. %B %e.", $this->szuldatum->getTimestamp()); }
    }
  }
?>
```

mysqli/Adatbazis.php

```
<?php
  class Adatbazis {
    private $mysqli = null;
    const SZERVER = "xenia.sze.hu";
    const FELHASZNALO = "hallgato";
    const JELSZO = "mysql";
    const ADATBAZIS = "hallgatok";
    const TABLA = "hallgatok";

    public function __construct() {
      @$kapcs = new mysqli(self::SZERVER, self::FELHASZNALO,
                           self::JELSZO, self::ADATBAZIS);
      if($kapcs->connect_error) {
        die("Nem sikerült megnyitni az adatbázist. Hibakód: ".
          $kapcs->connect_errno.", hibaüzenet: ".$kapcs->connect_error);
      } else {
        $this->mysqli = $kapcs;
      }
    }
```

mysqli/Adatbazis.php

```
public function __destruct() {
  if($this->mysqli != null) {
    $this->mysqli->close();
    $this->mysqli = null;
  }
}
protected function hiba($u) {
  print("<p>$u<br />\nMySQL hibakód: {$this->mysqli->errno}, ".
        "MySQL hibaüzenet: {$this->mysqli->error}</p>\n");
}
public function __toString() {
  $s = "";
  if($eredmeny = $this->mysqli->query("select * from ".self::TABLA.
     " order by nev")) {
    while($sor = $eredmeny->fetch_object("Hallgato")) {
      $s.="<p>$sor</p>\n";
    }
  } else $this->hiba("Hiba a hallgatók lekérdezésénél.");
  return $s; }
```

mysqli/Adatbazis.php

```
public function beszur(Hallgato $h) {
  if($kif = $this->mysqli->prepare("insert into ".self::TABLA.
     " values (?, ?, ?)")) {
    $k=$h->getNeptun(); $n=$h->getNev(); $d=$h->getSzuldatum()->
    format("Y-m-d"); $kif->bind_param("sss", $k, $n, $d);
    if(!$kif->execute()  || $kif->affected_rows == 0) {
      $this->hiba("Nem sikerült az új objektum beszúrása.");
    }
    $kif->close();
  } else {
    $this->hiba("Nem sikerült létrehozni az SQL kifejezést.");
  }
}
public function csere(Hallgato $h) {
  if(!$this->mysqli->query("update ".self::TABLA." set nev=".
    "\"{$h->getNev()}\", szuldatum=\"{$h->getSzuldatum()->".
    "format("Y-m-d")}\" where neptun=\"{$h->getNeptun()}\"")) {
    $this->hiba("Nem sikerült az objektum módosítása.");
  } elseif($this->mysqli->affected_rows == 0) {
    $this->hiba("Ismeretlen hallgató, nem módosítható."); }
}
```

mysqli/Adatbazis.php

```
    public function torles(Hallgato $h) {
      if(!$this->mysqli->query("delete from ".self::TABLA.
        " where neptun=\"{$h->getNeptun()}\"")) {
        $this->hiba("Nem sikerült az objektum törlése.");
      } elseif($this->mysqli->affected_rows == 0) {
        $this->hiba("Ismeretlen hallgató, nem törölhető.");
      }
    }
  }
?>
```

PDO

PDO: PHP Data Objects

- adatelérési absztrakciós réteg, minden AB szerverhez külön meghajtó szükséges
- pehelysúlyú, konzisztens, PHP5.0+ szükséges az OO szemlélete miatt

Kapcsolódás a szerverhez

```
public PDO::__construct ( string $dsn [, string $username
                        [, string $password [, array $options ]]] )
```

dsn Data Source Name

1. meghajtónév : kulcs1=ertek1; kulcs2=ertek2
2. URI a fájlra, ami tartalmazza a DSN-t
3. a php.ini-ben definiált pdo.dsn.azonosító-ra mutató álnév

username felhasználónév

password jelszó

options meghajtó-specifikus kulcs-érték párok, pl. MySQL-nél PDO::ATTR_PERSISTENT

PDO

Hibakezelés: PDOException

- öröklött: protected int $code ;
- saját: protected string $code ;

Hibakezelési mód változtatása: PDO::setAttribute(),
PDO::ATTR_ERRMODE értékei:

PDO::ERRMODE_SILENT csak hibakódokat állít be a PDO és PDOStatement objektumokban → errorCode(), errorInfo()

PDO::ERRMODE_WARNING E_WARNING üzenetet is generál (fejlesztés/teszt)

PDO::ERRMODE_EXCEPTION PDOException kivételt vált ki

PDO

SQL utasítások végrehajtása: PDO::query()

- egy kötelező paramétere van, az SQL string
- további paraméterekkel a visszaadott PDOStatement obj. feldolgozási módja állítható

PDOStatement

- Traversable interfész → foreach
- setFetchMode() lekérési üzemmód (aé.: oszlopnevekkel indexelt tömb), pl.:

PDO::FETCH_NUM	számokkal indexelt tömb
PDO::FETCH_ASSOC	oszlopnevekkel indexelt tömb
PDO::FETCH_CLASS	a köv. paraméterként adott nevű osztályt példányosítja (nem a konstruktorral)
PDO::FETCH_INTO	létező objektumot tölt fel adatokkal

További PDOStatement metódusok

- PDOStatement::fetch() eredménytábla aktuális sorának lekérése, paraméterként adott módon (PDO::FETCH_*) és irányban bejárva, kezdősortól
- PDOStatement::fetchObject() lekérés objektumonként
- PDOStatement::rowCount() INSERT/UPDATE/DELETE által érintett sorok száma, néhány AB az eredménytábla sorainak számát is megadja

Előkészített kifejezések

- Sablon megadása: PDO::prepare(), helyőrzők:
  - :cimke formában (kifejezőbb)
  - ? karakterekkel (indexelés 1-től kezdve)
- vissza: PDOStatement
- Helyettesítési értékek bekötése:
  - PDOStatement::bindValue() értéket köt be
  - PDOStatement::bindParam() változót (referencia) köt be
- paraméterek: címke, érték/változó, adattípus (pl. PDO::PARAM_STR)
- Adatcsomag küldése: PDOStatement::execute(), ennek is átadhatóak a helyettesítendő értékek tömbként
- Utoljára beszúrt sor ID-je: PDO::lastInsertId()

pdo/Adatbazis.php

```
<?php
  declare(strict_types = 1);
  class Adatbazis {
    private $dbh = null;
    const SZERVER = "xenia.sze.hu";
    const FELHASZNALO = "hallgato";
    const JELSZO = "mysql";
    const ADATBAZIS = "hallgatok";
    const TABLA = "hallgatok";

    public function __construct() {
      try {
        $this->dbh = new PDO("mysql:host=".self::SZERVER.
          ";dbname=".self::ADATBAZIS, self::FELHASZNALO, self::JELSZO);
        $this->dbh->setAttribute(PDO::ATTR_ERRMODE,
                                 PDO::ERRMODE_EXCEPTION);
      } catch(PDOException $e) {
        die("Nem sikerült megnyitni az adatbázist. Hibakód: ".
            $e->getCode().", hibaüzenet: ".$e->getMessage());
      } }
```

pdo/Adatbazis.php

```
public function __destruct() {
  $this->dbh = null;
}
protected function hiba($u, $e) {
  print("<p>$u<br />\nPDO hibakód: ".$e->getCode().
        ", PDO hibaüzenet: ".$e->getMessage()."</p>\n");
}
public function __toString() :string {
  $s = "";
  try {
    $eredmeny = $this->dbh->query("select * from ".self::TABLA.
                                  " order by nev");
    foreach($eredmeny as $sor) {
      $hg = new Hallgato($sor["nev"], $sor["neptun"],
                         new DateTime($sor["szuldatum"]));
      $s.="<p>$hg</p>\n";
    }
  } catch(PDOException $e) {
    $this->hiba("Nem sikerült lekérdezni a hallgatók adatait.", $e);
  }
  return $s; }
```

pdo/Adatbazis.php

```
public function beszur(Hallgato $h) {
  try {
    $kif = $this->dbh->prepare("insert into ".self::TABLA.
       " values (:neptun ,:nev, :szuldatum)");
    $kif->bindValue(":neptun", $h->getNeptun());
    $kif->bindValue(":nev", $h->getNev());
    $kif->bindValue(":szuldatum",
      $h->getSzuldatum()->format("Y-m-d"));
    $kif->execute();
  } catch(PDOException $e) {
    $this->hiba("Nem sikerült az új objektum ".
                "adatbázisba illesztése.", $e);
  }
}
```

pdo/Adatbazis.php

```
public function csere(Hallgato $h) {
  try {
    $kif = $this->dbh->prepare("update ".self::TABLA.
      " set nev=?, szuldatum=? where neptun=?");
    $k=$h->getNeptun(); $n=$h->getNev();
    $d=$h->getSzuldatum()->format("Y-m-d");
    $kif->bindParam(1, $n, PDO::PARAM_STR);
    $kif->bindParam(2, $d, PDO::PARAM_STR);
    $kif->bindParam(3, $k, PDO::PARAM_STR, 6);
    $kif->execute();
    if($kif->rowCount() == 0)
      throw new PDOException("Ismeretlen Neptun kód.");
  } catch(PDOException $e) {
    $this->hiba("Nem sikerült az objektum módosítása.", $e);
  }
}
```

pdo/Adatbazis.php

```
    public function torles(Hallgato $h) {
      try {
        $eredmeny = $this->dbh->query("delete from ".self::TABLA.
          " where neptun=\"{$h->getNeptun()}\"");
        if($eredmeny->rowCount() == 0)
          throw new PDOException("Ismeretlen Neptun kód.");
      } catch(PDOException $e) {
        $this->hiba("Nem sikerült az objektum törlése.", $e);
      }
    }
  }
?>
```

Doctrine2

A Doctrine főbb tulajdonságai

- ORM = Object Relational Mapper
- a Java Persistence API ihlette
- az üzleti logikát leválasztja az adattárolási rétegről
- a korábbi, 1.x-es sorozathoz képest akár 3x-os sebességnövekedés
- a tárolandó objektumokat nem kell származtatni más osztályból
- bármilyen adattag menthető

Mire lesz szükség?

- PHP 5.4+
- Doctrine\Common: minden Doctrine projekt közös utility-jei
- Doctrine\DBAL: DataBase Abstraction Layer
- Symfony\Component\Console: CLI futtatóeszköz
- Symfony\Component\Yaml: csak, ha YAML konfigurációs fájlt szeretnénk használni

Composer

- projekt-szintű csomagkezelés, függőségek vizsgálata
- telepíthető csomagok listája
- első lépés: projekt mappa létrehozása, Composer beszerzése

Telepítés Composerrel (parancssorból)

```
wajzy@xenia:~$ uname -a
Linux xenia 3.16.0-4-686-pae #1 SMP Debian 3.16.36-1+deb8u1
(2016-09-03) i686 GNU/Linux
wajzy@xenia:~$ cd www
wajzy@xenia:~/www$ mkdir neptun
wajzy@xenia:~/www$ cd neptun
wajzy@xenia:~/www/neptun$ curl -s https://getcomposer.org/installer | php
wajzy@xenia:~/www/neptun$ sudo mv composer.phar /usr/local/bin/composer
wajzy@xenia:~/www/neptun$ mcedit composer.json
```

Második lépés: Composer konfigurálása

composer.json

```
{
    "require": {
        "doctrine/orm": "2.5.*"
    },
    "autoload": {
        "psr-4": {"": "src/"}
    }
}
```

Konfiguráció jelentése:

- require: a megjelölt csomagok és verziószámok a projektünk függőségei
- autoload: a használandó PHP osztálybetöltő típusa; PSR-4 néha problémás
  - PSR = PHP Standards Recommendation
  - PSR-0: elavult
  - PSR-4 általában javasolt megoldás
  - a projekt src mappájában keresi majd a betöltő a forrásainkat

(Doctrine 2.6.* → PHP7.1+; telepítési útmutató)

Harmadik lépés: függőségek telepítése Composerrel

Csomagok telepítése

```
wajzy@xenia:~/www/neptun$ composer install
```

Negyedik lépés: az entitás elkészítése

- Entity: tárolandó adatot tartalmazó PHP osztály/objektum
- nem kell semmiből származtatni őket/interfészt megvalósítani
- nem lehet final, nem tartalmazhat final metódusokat
- általában nem implementálja a __clone() és __wakeup() metódusokat, de ha mégis kellene...

Entitás előkészítése

```
wajzy@xenia:~/www/neptun$ mkdir src
wajzy@xenia:~/www/neptun$ cd src
wajzy@xenia:~/www/neptun/src$ mcedit Hallgato.php
```

src/Hallgato.php

```php
<?php
  declare(strict_types = 1);

  /**
    * @Entity
    * @Table(name="hallgatok")
    */
  class Hallgato {
    /** @Id
      * @Column(type="string", length=6)
      * @var string
      */
    private $neptun;

    /**
      * @Column(type="string")
      * @var string
      */
    private $nev;
```

src/Hallgato.php

```
/**
  * @Column(type="datetime")
  * @var DateTime
  */
private $szuldatum;

public function __construct(string $neptun,
                            string $nev,
                            DateTime $szuldatum) {
  $this->neptun = $neptun;
  $this->nev = $nev;
  $this->szuldatum = $szuldatum;
}

public function getNeptun() :string {
  return $this->neptun;
}
```

Magyarázat

- A tárolandó adatokról PHP annotációkkal rendelkezünk, de használhatnánk XML-t és YAML-t is (ki kellene egészíteni hozzá a composer.json require kulcshoz tartozó objektumát a "symfony/yaml": "2.5.*" adattaggal)
- Annotáció referencia
- @Entity(repositoryClass="MyProject\UserRepository") entitásként jelöli meg a PHP osztályt. `repositoryClass` attribútum: opcionális, az alapértelmezett repository osztály helyett sajátot szeretnénk használni, és ebben a modelltől elválasztva foglaljuk össze az entitáshoz kapcsolódó DQL/SQL utasításokat.
- @Table(name="user") a `name` kötelező attribútummal adott táblában fogja tárolni az entitást

Doctrine2

- @Id egyedi azonosító, elsődleges kulcs
- @GeneratedValue(strategy="IDENTITY") csak @Id-vel használható együtt, a `strategy` attribútum elhagyható, alapértelmezetten `AUTO` értékű (az AB állítja elő az egyedi azonosítót, pl. AUTO_INCREMENT a MySQL-nél). Az annotáció hiánya a `NONE` érték szinonimája.
- @Column megjelöli az adattagot tárolásra. `type` kötelező attribútum, Doctrine típus a PHP/AB közti konverzióhoz. `length` opcionális attribútum, string-ek maximális hossza (ld. varchar). `unique`, opcionális, egyedi értéket követel meg. `nullable`, opcionális, meghatározza, hogy NULL érték elfogadható-e.
- @var nem Doctrine annotáció, az adattag elvárt típusát adja meg.
- A Doctrine nem a konstruktort használja az adattagok feltöltésére.

Doctrine2

- Az egyedi azonosítóhoz (@Id) jellemzően nem tartozik set metódus.
- Többnyire getter/setter nélkül is eléri az adattagokat (ld. Reflection API), de néha szükséges lehet definiálni ilyeneket.
- az annotációkhoz osztályok tartoznak, ezeket be kell tölteni (ld. később)

src/Hallgato.php

```
    public function getNev() : string {
      return $this->nev;
    }
    public function setNev(string $nev) {
      $this->nev = $nev;
    }
    public function getSzuldatum() : DateTime {
      return $this->szuldatum;
    }
    public function setSzulDatum(DateTime $datum) {
      $this->szuldatum = $datum;
    }
    public function __toString() : string {
      return "Hallgató neve: {$this->nev}, ".
        "neptun kódja: {$this->neptun}, ".
        "születési dátuma: ".
        strftime("%Y. %B %e.", $this->szuldatum->getTimestamp());
    }
  }
?>
```

Ötödik lépés: az EntityManager megszerzése

- ezen keresztül érhetők el a Doctrine szolgáltatásai

bootstrap.php

```
<?php
use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;

require_once "vendor/autoload.php";

$isDevMode = true;
$config = Setup::createAnnotationMetadataConfiguration(
  array(__DIR__."/src"), $isDevMode);

$conn = array(
    'driver'   => 'pdo_mysql',
    'user'     => 'hallgato',
    'password' => 'mysql',
    'dbname'   => 'hallgatok'
);

$entityManager = EntityManager::create($conn, $config);
```

Magyarázat

- `require_once`: aktiválja a Composer automatikus osztálybetöltőjét
- `$isDevMode = true;` a gyorsítótárazást és a proxy osztályok használatát befolyásolja
- A `Setup` egy segédosztály, amivel most a konfigurációs objektumot kérjük le
- `createAnnotationMetadataConfiguration` első paramétere a forráskönyvtárunk
- XML: `createXMLMetadataConfiguration(array(__DIR__."/config/xml"), $isDevMode);`
- YAML: `createYAMLMetadataConfiguration(array(__DIR__."/config/yaml"), $isDevMode);`
- A `$conn` tömb tartalmazza az AB eléréséhez szükséges adatokat. Nem csak MySQL-lel működik.
- Végül egy factory metódus megadja az `EntityManagert`.

Hatodik lépés: az adatbázis séma létrehozása

- SchemaTool parancssoros utility-vel fogjuk elvégezni
- szüksége van egy segédfájlra

cli-config.php

```
<?php
require_once "bootstrap.php";

return \Doctrine\ORM\Tools\Console\ConsoleRunner::
       createHelperSet($entityManager);
```

SchemaTool futtatása

```
wajzy@xenia:~/www/neptun$ vendor/bin/doctrine orm:schema-tool:create
```

Opcionális: séma törlése, ismételt elkészítése

```
wajzy@xenia:~/www/neptun$ vendor/bin/doctrine orm:schema-tool:drop --force
wajzy@xenia:~/www/neptun$ vendor/bin/doctrine orm:schema-tool:create
```

Opcionális: séma frissítése

```
wajzy@xenia:~/www/neptun$ vendor/bin/doctrine orm:schema-tool:update --force
```

Hetedik lépés: további forrásfájlok elkészítése

src/Evfolyam.php

```
<?php
  declare(strict_types = 1);

  use Doctrine\ORM\EntityManager;

  class Evfolyam {
    private $em;

    public function __construct(EntityManager $em) {
      $this->em = $em;
    }

    public function urlap() { ... }
```

src/Evfolyam.php

```
public function feldolgozas() {
  if(isset($_POST["mod"]) &&
    in_array($_POST["mod"], ['beszur', 'csere', 'torles'])) {
    try {
      switch($_POST["mod"]) {
        case 'beszur':
          $hg = new Hallgato($_POST["neptun"], $_POST["nev"],
                             new DateTime($_POST["szuldatum"]));
          $this->em->persist($hg);
          break;
        case 'csere':
          $hg = $this->em->find('Hallgato', $_POST["neptun"]);
          if ($hg === null) {
            throw new Exception();
          }
          $hg->setNev($_POST["nev"]);
          $hg->setSzulDatum(new DateTime($_POST["szuldatum"]));
          break;
```

src/Evfolyam.php

```
            case 'torles':
              $hg = $this->em->find('Hallgato', $_POST["neptun"]);
              if ($hg === null) {
                throw new Exception();
              }
              $this->em->remove($hg);
              break;
          }
          $this->em->flush();
        } catch(Exception $e) {
          echo "<p style=\"color: red\">{$_POST["neptun"]} ".
            "kódú hallgató nem létezik.</p>\n";
        }
      }
    }
```

Magyarázat

- `persist()` új objektumok tárolását kéri.
- `find()`, egy objektum kikeresése egyedi azonosító alapján.
- `remove()` objektum eltávolítása.
- Tényleges írás `flush()` hatására (tranzakciókezelés, teljesítmény fokozása).

src/Evfolyam.php

```
    public function lista() {
      echo "<h1>A hallgatók listája:</h1>\n";
      $hallgatoRepository = $this->em->getRepository('Hallgato');
      $hallgatok = $hallgatoRepository->findAll();

      foreach ($hallgatok as $hg) {
        echo '<p>', $hg, "</p>\n";
      }
    }
  }
?>
```

Magyarázat

- `getRepository()`: alapértelmezett repository elkérése, amiben olyan kereső metódusok vannak, mint pl.
- `findAll()`: összes objektum visszaadása

index.php

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Doctrine2 ORM</title>
  </head>
  <body>
    <?php
      setlocale(LC_ALL, "hu_HU.UTF-8");

      require_once "bootstrap.php";

      $ef = new Evfolyam($entityManager);
      $ef->feldolgozas();
      $ef->lista();
      $ef->urlap();
    ?>
  </body>
</html>
```

Továbbfejlesztés: egy hallgató → több elérhetőség

- Egy a többhöz, kétirányú asszociációs kapcsolat
- Lehetne még: egyirányú kapcsolat kapcsolótáblával
- vagy persze egy sima PHP tömbbel, de ahhoz nem kellene újabb AB tábla ;)

src/Hallgato.php

```
<?php
  declare(strict_types = 1);

  use Doctrine\Common\Collections\ArrayCollection;
  ...
  /**
    * @OneToMany(targetEntity="Elerhetoseg",
                 mappedBy="hallgato_neptun",
                 cascade={"persist", "remove"})
    */
  private $elerhetosegek = null;
```

src/Hallgato.php

```
public function __construct(string $neptun,
                            string $nev,
                            DateTime $szuldatum) {
  $this->neptun = $neptun;
  $this->nev = $nev;
  $this->szuldatum = $szuldatum;
  $this->elerhetosegek = new ArrayCollection();
}
...
public function __toString() : string {
  $s = "Hallgató neve: {$this->nev}, ".
       "neptun kódja: {$this->neptun}, ".
       "születési dátuma: ".
       strftime("%Y. %B %e.", $this->szuldatum->getTimestamp()).
       "\n<ul>\n";
  foreach($this->elerhetosegek as $elerhetoseg) {
    $s .= "<li>$elerhetoseg</li>\n";
  }
  $s .= "</ul>\n";
  return $s;
}
```

src/Hallgato.php

```
    public function clearElerhetosegek() {
      $this->elerhetosegek->clear();
    }

    public function addElerhetosegek(string $elerhetosegek) {
      $lista = explode(",", $elerhetosegek);
      foreach($lista as $elem) {
        $e = new Elerhetoseg();
        $e->setElerhetoseg(trim($elem));
        $e->setNeptun($this);
        $this->elerhetosegek[] = $e;
      }
    }
  }
?>
```

Magyarázat:

- ArrayCollection becsomagol egy PHP tömböt, hogy nyomon követhesse tartalmának változását
- @OneToMany jelzi, hogy az aktuális osztály egy példányához több is tartozhat a másikból
  - `targetEntity`: a másik entitás, amivel össze akarunk kapcsolódni
  - `mappedBy`: a másik entitás adattagja, ami ezt az objektumot fogja címezni
  - `cascade`: ha az aktuális objektummal műveletet végzek, a kapcsolt objektumokkal ne kelljen explicit módon foglalkoznom
- az `Elerhetoseg` objektumnak expliciten hívni kell a `setNeptun()` metódusát, különben az AB-ba NULL értékek kerülnek, és az elérhetőségek nem törlődnek a táblából (nem lehet egymáshoz kapcsolni az objektumokat)

src/Elerhetoseg.php

```
<?php
  declare(strict_types = 1);
  /**
    * @Entity
    * @Table(name="elerhetosegek")
    */
  class Elerhetoseg {
    /**
      * @Id
      * @GeneratedValue
      * @Column(type="integer")
      * @var int
      */
    protected $id;

    /**
      * @ManyToOne(targetEntity="Hallgato", inversedBy="neptun")
      * @JoinColumn(name="hallgato_neptun",
                    referencedColumnName="neptun")
      */
    protected $hallgato_neptun;
```

src/Elerhetoseg.php

```
/**
  * @Column(type="string")
  * @var string
  */
protected $elerhetoseg;

/**
  * @return int
  */
public function getId() : int {
  return $this->id;
}

/**
  * @return string
  */
public function getElerhetoseg() : string {
  return $this->elerhetoseg;
}
```

src/Elerhetoseg.php

```
    /**
      * @param string $elerhetoseg
      */
    public function setElerhetoseg(string $elerhetoseg) {
      $this->elerhetoseg = $elerhetoseg;
    }

    public function setNeptun(Hallgato $hallgato) {
      $this->hallgato_neptun = $hallgato;
    }

    public function __toString() {
      return $this->elerhetoseg;
    }
  }
```

Magyarázat:

- @ManyToOne Ez a „több” oldal
  - `targetEntity` az „egy” oldal entitása
  - `inversedBy` az „egy” oldal egyedi azonosítója, a `$hallgato_neptun` adattag értéke (idegen kulcs)
- @JoinColumn mivel nem az alapértelmezett azonosítót használom az „egy” oldalon (`$id` helyett `$neptun` adattag), expliciten kell összekapcsolni a két oldalt egymással

Adatbázis újjászervezése

```
wajzy@xenia:~/www/neptun$ vendor/bin/doctrine orm:schema-tool:update --force --dump-sql
CREATE TABLE elerhetosegek (id INT AUTO_INCREMENT NOT NULL, hallgato_neptun VARCHAR(6)
DEFAULT NULL, elerhetoseg VARCHAR(255) NOT NULL, INDEX IDX_4102C2F467DDC47E
(hallgato_neptun), PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci
ENGINE = InnoDB;
ALTER TABLE elerhetosegek ADD CONSTRAINT FK_4102C2F467DDC47E FOREIGN KEY (hallgato_neptun)
REFERENCES hallgatok (neptun);

Updating database schema...
Database schema updated successfully! "2" queries were executed
```

src/Evfolyam.php

```
public function urlap() {
...
  <label for="szuldatum">Születési dátum:</label>
    <input type="text" id="szuldatum" name="szuldatum" /><br />
  <label for="elerhetosegek">Elérhetőségek:</label>
    <input type="text" id="elerhetosegek"
           name="elerhetosegek" /><br />
  </fieldset>
  <p><input type="submit" value="OK" /></p>
</form>
...
public function feldolgozas() {
  if(isset($_POST["mod"]) &&
    in_array($_POST["mod"], ['beszur', 'csere', 'torles'])) {
    try {
      switch($_POST["mod"]) {
        case 'beszur':
          $hg = new Hallgato($_POST["neptun"], $_POST["nev"],
                             new DateTime($_POST["szuldatum"]));
          $hg->addElerhetosegek($_POST["elerhetosegek"]);
          $this->em->persist($hg);
          break;
```

src/Evfolyam.php

```
        case 'csere':
          $hg = $this->em->find('Hallgato', $_POST["neptun"]);
          if ($hg === null) {
            throw new Exception("Módosítás");
          }
          $hg->setNev($_POST["nev"]);
          $hg->setSzulDatum(new DateTime($_POST["szuldatum"]));
          $hg->clearElerhetosegek();
          $hg->addElerhetosegek($_POST["elerhetosegek"]);
          break;
        case 'torles':
          $hg = $this->em->find('Hallgato', $_POST["neptun"]);
          if ($hg === null) {
            throw new Exception("Törlés");
          }
          $this->em->remove($hg);
          break;
      }
      $this->em->flush();
```

src/Evfolyam.php

```
        } catch(Exception $e) {
          echo "<p style=\"color: red\">{$e->getMessage()}: ".
               "{$_POST["neptun"]} kódú hallgató nem létezik.</p>\n";
        }
      }
    }

    public function lista() {
      echo "<h1>A hallgatók listája:</h1>\n";
      $hallgatoRepository = $this->em->getRepository('Hallgato');
      $hallgatok = $hallgatoRepository->findAll();

      foreach ($hallgatok as $hg) {
        echo '<p>', $hg, "</p>\n";
      }
    }
  }
?>
```

Feladatok

Hozzon létre egy **Valasz** osztályt, ami egységbe zárja egy kérdésre adható lehetséges válasz szövegét, valamint azt, hogy ez a válasz igaz-e vagy hamis. Hozzon létre olyan metódust, ami paraméterként megkapja a felhasználó feleletét, majd visszaadja a pontszámot, ami erre a feleletre jár. Helyes feleletre 1, helytelenre -1 pont jár, illetve ha nem válaszolt a felhasználó a kérdésre, akkor 0.

Hozzon létre **Kerdes** osztályt, ami egy kérdés szövegét tárolja, illetve tetszőleges számú **Valasz** objektumot is tud aggregálni. Hozzon létre olyan metódust, ami a felhasználónak a kérdésre adott feleleteit kapja paraméterként, visszatérési értéke pedig a feleletekre járó pontok összege.

Hozzon létre **Teszt** osztályt, ami **Kerdes** objektumokat tárol, ezek közül egyet-egyet képes megjeleníteni egy űrlap formájában, és a felhasználó által visszaküldött űrlap adatok alapján képes a kérdésekre adott feleletek pontszámainak összegzésére.

Készítsen olyan programot, ami legalább két kérdést tesz fel a felhasználónak egymás után! A kérdés szövege után egymás alatt jelenítse meg a lehetséges válaszokat! Minden válasznál helyezzen el vezérlőket, melyekkel a felhasználó *igen*, *nem* és *nem tudom* típusú feleleteket tud adni a feltett kérdésekre. Számolja össze és jelenítse meg végül az összpontszámot!

Felhasznált irodalom

Antonio Lopez
Learning PHP7
Pact Publishing Ltd., Birmingham, UK, 2016.

Hivatalos PHP referencia

Doctrine tutorial

Marco Pivetta ORM tutorial