

Anleitung/Tutorium zur Erstellung einer objektrelationalen Abbildungsschicht (inkl. webbasierter Oberfläche)

• Einleitung

In den folgenden Lektionen soll die Realisierung einer objektrelationalen Abbildungsschicht gezeigt werden. Die Realisierung erfolgt mittels PHP, als zugrundeliegende Datenbank wird ein Auszug aus der Mondial Datenbank verwendet.

Benutze dazu deinen Oracle Account an der FH oder einen beliebigen Account auf einer Oracle Installation bei dir Zuhause.

Führe die im folgenden aufgeführten Aufgaben der Reihe nach durch. Falls du Probleme hast, wirf zuerst mal einen Blick in den Teil „Allgemeine Hinweise“, da findet du ein paar hilfreiche Tipps. Die Übung wird nicht bewertet.

Im folgenden soll jetzt eine OR-Schicht realisiert werden, die auf einer Teilmenge der Tabellen und Attribute der Original Mondial Datenbank basiert. Im einzelnen sollen folgende Tabellen und Attribute verwendet werden:

Tabelle CITY: (NAME, COUNTRY, PROVINCE, POPULATION)

Tabelle COUNTRY (NAME, CODE, CAPITAL, PROVINCE, POPULATION)

Tabelle BORDERS (COUNTRY1, COUNTRY2)

Vorbereitende Arbeiten:

Kopiere die entsprechenden Attribute der Tabellen MONDIAL.COUNTRY, MONDIAL.CITY und MONDIAL.BORDERS mit ihren Inhalten in dein Schema. Setze weiterhin die Primär- und Fremdschlüssel, so wie sie auch im Mondial Schema definiert wurden (Informationen darüber findest du in den Tabellen ALL_CONSTRAINTS und ALL_CONS_COLUMNS).

Zugriff von PHP auf die Datenbank:

In Anhang A ist eine Klasse, welche den Zugriff auf die Datenbank erleichtert abgedruckt. Kopier diese Klasse in eine Datei mit dem Namen *DB_Util.php* und speichere sie ab. Weiterhin findest du in Anhang B eine kleine Testdatei, welche diese Klasse nutzt. Speichere den Code aus Anhang B ebenfalls in einer Datei (Name: *mondial-access.php*) ab und führe fol-

(Version 0.3 vom 1.6.2006)

gendes Statement (in CYGWIN) aus:

```
$ h:/Schmidt/xampp/php/php.exe mondial-access.php
```

Modifiziere das Skript so, dass für jedes Land der höchste Berg ausgegeben wird.

Modifiziere das Skript weiterhin so, dass du auf deinen Datenbankaccount zugreifst und Hinterweidenthal (Pfalz) in die Datenbank einträgst. Lies den Ort anschließend zur Kontrolle wieder aus.

Ok, genug gespielt, weiter gehts ;-)

Ermittlung der Struktur der Tabellen

Mittels der Kommandos

DESCRIBE CITY

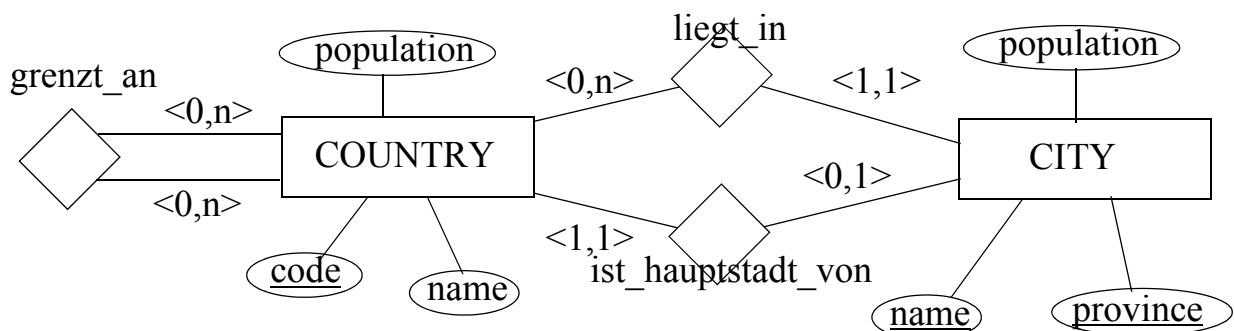
DESCRIBE COUNTRY

DESCRIBE BORDERS

kann die Struktur der Tabellen ermittelt werden.

Die Beziehungen lassen sich mittels den entsprechenden Einträgen in den Tabellen ALL_CONSTRAINTS und ALL_CONS_COLUMNS ermitteln.

Daraus läßt sich das ER-Modell des Schemas ableiten, das wiefolgt aussieht:



Erläuterungen: Primärschlüssel sind unterstrichen. Die Attribute *province* und *capital* der Tabelle *COUNTRY* sind Umsetzungen der Beziehung *ist_hauptstadt_von* des ER-Modells auf das relationale Modell und tauchen deshalb im ER-Modell auch nicht auf. Analog verhält es sich mit dem Attribut *country* der Tabelle *CITY*.

Die n:m-Beziehung *grenzt_an* wird im relationalen Modell mittels der Beziehungstabelle *BORDERS* realisiert, aus diesem Grund tauchen auch die beiden Attribute *COUNTRY1* und *COUNTRY2* im ER-Modell nicht auf.

Aus dem ER-Modell lassen sich jetzt eine Reihe von Eigenschaften für die objektrelationale Abbildungsschicht ableiten:

1.) Entitäten werden als Klassen realisiert:

=> Klasse *Country*, Klasse *City*.

Die Klasse muss Methoden zum Anlegen von Instanzen in der Datenbank, zum Löschen und Modifizieren haben. Weiterhin sind Methoden zum Auffinden einzelner Datensätze (*get_by_primary_key(<id>)*) und Auslesen mehrerer Datensätze entsprechend einer Anfragebedingung notwendig (*query(<condition>)*).

Übung: Trage in die untenstehende Tabelle die notwendigen Methoden ein und sage ob es sich dabei um eine Klassen- oder Instanzenmethode handelt. Gib weiterhin Rückgabewert und Parameter an.

Tabelle 1:

	Methodenname	Ergebnistyp	Parameter	Beschreibung
Klasse City				

Tabelle 1:

	Methodenname	Ergebnistyp	Parameter	Beschreibung
Klasse Country				

2.) Beziehungen können mittels Methoden realisiert werden¹. Um die Methoden genauer spezifizieren zu können, betrachtet man die Kardinalitäten:

Eine $\langle 0, 1 \rangle$ oder $\langle 1, 1 \rangle$ Kardinalität wie bei der Beziehung *liegt_in* auf Seite der Entität *CITY* bedeutet, dass eine Stadt in genau (Minimalkardinalität 1) bzw. maximal (Maximalkardinalität 1) einem Land liegen kann. Das bedeutet, dass eine Methode (*get_country()*) der Klasse *City*, welche uns das entsprechende Land zurückliefern soll, liefert genau eine Instanz eines Landes (oder null) zurückliefert. Soll das Land hingegen gesetzt werden, so wird eine Methode *set_country(<country>)* benötigt, welche das Land für diese Stadt setzt (um die Beziehung wieder zu lösen kann man entweder eine Methode *unset_country()* definieren oder man erlaubt die Übergabe des Null-Wertes an die Methode *set_country(<country>)*).

1. alternativ können Beziehungen durch zusätzliche Instanzenvariablen vom entsprechenden Typ (bei $\langle 0, 1 \rangle$ Kardinalitäten) oder mengenwertigen Instanzenvariablen (bei $\langle 0, 1, n \rangle$ Kardinalitäten) abgebildet werden.

Anders verhält sich der Fall bei $\langle 0, n \rangle$ und $\langle 1, n \rangle$ Kardinalitäten. Betrachten wir die andere Seite der Beziehung *liegt_in*. Diese sagt aus, dass in einem Land zwischen 0 und n-Städte liegen können. Eine Methode, welche alle Städte eines Landes zurückliefern soll muss also eine Menge von Instanzen vom Typ *City* zurückliefern (in einem Array). Entsprechend werden Methoden *add_city*(*<city>*), *delete_city*(*<city>*) und eventuell *set_cities* (*<array of cities>*) benötigt um die Beziehung verwalten zu können.

Übung: Betrachte alle Beziehungen des ER-Modells und trage in untenstehende Tabelle die notwendigen Methoden für die beiden Klassen ein:

Tabelle 2:

	Methodenname	Ergebnistyp	Parameter	Beschreibung
Klasse City				
Klasse Country				

Tabelle 2:

	Methodenname	Ergebnistyp	Parameter	Beschreibung

Erstelle in einem Editor folgende Klassendefinition:

```
<?
```

```
include_once 'DB_Util.php';
```

```
class Country {
```

```
    private $code;
```

```
    private $name;
```

```
    private $population;
```

```
    private $capital;
```

```
    private $province;
```

```
    function __construct($code, $name, $population=null,  
                        $capital=null, $province=null) {
```

```
        $this->code = $code;
```

```
        $this->name = $name;
```

```
        $this->population = $population;
```

```
        $this->capital = $capital;
```

```
        $this->province = $province;
```

```
}

function __toString() {
    return "$this->name ($this->code) [Einwohner: $this->population,
Hauptstadt: $this->capital ($this->province)]";
}

static function get($code) {
    $country = null;
    $list = DB_Util::query("
        select code, name, population, capital, province
        from mondial.country
        where code= ? ", $code);
    if (count($list)> 0) {
        $res = $list[0];
        $country = new Country($res['CODE'], $res['NAME'],
            $res['POPULATION'],
            $res['CAPITAL'], $res['PROVINCE']);
    }
    return $country;
}
}

DB_Util::connect("oci8://dein_account/dein_passwort@widb1");

echo Country::get('D');

DB_Util::close();

?>
```

Idee hierbei ist, für jede Spalte in der Tabelle eine entsprechende Instanzenvariable (*\$name*, *\$code*, ...) zu definieren.

Die Klasse verfügt über eine Instanzenmethode *__toString()*, welche eine textuelle Repräsentation der Instanz zurückliefert.

Die statische Methode *get(\$code)* zeigt exemplarisch, wie ein konkreter Datensatz aus der Mondial-DB ausgelesen wird und als Instanz der Klasse *Country* zurückgeliefert wird.

Aufgaben:

- Erweitere die Datei nun analog um die Klasse *City*.
- Erstelle in der Klasse *Country* eine Instanzenmethode *get_cities()*, welche alle Städte dieses Landes als Instanzen vom Typ *City* zurückliefert (in einem Array).
- Modifiziere den Konstruktor der Klasse *Country* derart, dass er nur einen Parameter (vom Typ assoziativer Array) entgegennimmt, der die Informationen aller Felder beinhaltet.

Beispiel:

```
$dic['NAME'] = 'Baden Baden';  
$dic['POPULATION'] = 53421;  
$dic['COUNTRY'] = 'D';  
$nice_city = new City($dic);
```

Vorteile dieser Lösung sind, dass die Reihenfolge der Parameter keine Rolle spielt und dass das Übergabeformat genau mit dem übereinstimmt, das von der Methode

```
$res->fetchRow(DB_FETCHMODE_ASSOC)
```

beim Auslesen der Ergebnisse aus der Datenbank zurückgeliefert wird. Erstelle mit dieser Kenntnis nun eine weitere Methode

```
DB_Util::object_query($sql, $class, $paras),
```

die im Unterschied zur Methode *DB_Util::query(\$sql, \$paras)* Instanzen vom angegebenen Typ (Parameter *\$class*) zurückliefert.

Erstelle mithilfe der Methode *object_query(...)* eine Instanzenmethode *neighbours()*, welche für ein Land, alle angrenzenden Nachbarn zurückliefert (als Instanzen vom Typ *Country*).

- Erstelle für die beiden Klassen per hand die fehlenden getter-/ und setter-Methoden. Alternativ (aus Faulheit, Wissensdurst, Innovationsdrang, ...) kannst du dir die Funktionsweise der Magic-Methode *__call(...)* genauer anschauen¹.

Erstelle für die Klasse *Country* eine statische Methode *query(\$condition)*, die alle die Instanzen zurückliefert, welche die Bedingung *\$condition* erfüllen. Optional kann noch ein zweiter Parameter genutzt werden um eine Sortierreihenfolge zu definieren.

1. [http://www.phpbar.de/w/Call\(\)](http://www.phpbar.de/w/Call())

Als nächstes sollen zwei Views (PHP Seiten, die Inhalte aus dem Modell auslesen und anzeigen) realisiert werden. Der erste View soll alle Länder in Tabellenform auflisten, der zweite View soll Details über ein Land ausgeben.


Erstelle dazu ein Verzeichnis unterhalb des Verzeichnisses *htdocs* und kopiere die Dateien *DB_Util.php* (mit deinen Erweiterungen) und die Datei mit den Klassendefinitionen für *City* und *Country* hinein. Erstelle weiterhin eine Datei, welche den eben erstellten View enthält (d.h. die Tabelle ausgibt).

Anschließend soll ein weiterer View erstellt werden, der Details zu einem Land ausgeben soll. Dazu gehören u.a. die Städte, die in diesem Land liegen. Damit der View weiß, welches Land er anzeigen soll, muss dem View ein Parameter mit übergeben werden (in etwa so:

`http://localhost/tutorium/show_country_detail.php?code=D`).

Nachdem die beiden Views erstellt wurden können diese jetzt miteinander verbunden werden. Dazu müssen nur die Tabelleneinträge des ersten Views als Link realisiert werden, der dann (entsprechend parametrisiert) auf die Detailseite verweist.


Nächster und letzter Schritt: Erstellen eines Controllers:

Erweitere die Tabelle des ersten Views um eine zusätzliche Spalte, die ein  Zeichen enthält¹. Das Zeichen wird als Grafik wie folgt eingebunden:

``. Als nächstes verbinden wir das Zeichen mit einem Controller. Dazu bauen wir um die Grafik einen Link, der wie folgt aussieht:

```
<a href="controller_delete_country.php?code=laendercode" border="0">
    
</a>
```

laendercode muss dabei entsprechend mit der ID des jeweiligen Landes ersetzt werden.

Durch Drücken der Grafik() wird von nun an der Controller gestartet. Dieser soll nun im letzten Schritt realisiert werden. Erweitere dafür zuerst deine Zugriffsschicht um eine Instanzenmethode *Country::delete()*, welche für das Löschen der konkreten Instanz in der Datenbank verantwortlich ist.

1. Geh bei Googles Bildersuche stöbern

Der Controller hat dann schlussendlich folgende Struktur:

```
$code = $_REQUEST['code'];
$country = Country::get($code);
$country->delete();
header("location: $zielseite");
```

\$zielseite ist hierbei eine URL, die nach dem Löschen angezeigt werden soll (z.B. wieder der erste View, der das soeben gelöschte Land dann nicht mehr anzeigen dürfte).

Anhang A (DB_Util Klasse für Datenbankzugriff)

Datei: h:\schmidt\xampp\htdocs\wwwenv\or-mondial-example\DB_Util.php

```
<?
```

```
// Klasse mit statischen Methoden zur Verwaltung einer
// Datenbankverbindung.
// Einsatz:
//
//      $db = DB_Util::connect($dsn);
//      $db_handle = DB_Util::get_connection();
//      $list = DB_Util::query("select * from cat");
//      DB_Util::close();
//
```

```
include_once('DB.php'); // load PEAR::DB class
```

```
class DB_Util {

    private static $db_handle;

    static function connect($dsn = "mysql://root:@localhost/mondial") {
        $c = DB::connect($dsn);
        if (DB::isError($c))
            die ("Fehler beim Verbindungsaufbau mit [$dsn] : ".$c->getDebug-
            Info());
    }
}
```

```
        self::$db_handle = $c;
    }

    static function get_connection() {
        if (empty(self::$db_handle))
            die("es besteht keine offene Datenbankverbindung. Zuerst
mit DB_Util::connect(\$dsn) eine Verbindung herstellen.");
        return self::$db_handle;
    }

    static function query($sql, $data=array()) {
        $db = self::get_connection();
        $stmt = $db->prepare( $sql );
        if (DB::isError($stmt))
            die("<pre>$sql </pre>:". $stmt->getDebugInfo());
        $res = $db->execute($stmt, $data);
        if (DB::isError($res))
            die("execute: <pre>$sql</pre>:". $res->getDebugInfo());
        $results = array();
        if (is_Object($res)) {
            while($row = $res->fetchRow(DB_FETCHMODE_ASSOC))
                $results[] = $row;
            return $results;
        } else {
            return $res;
        }
    }

    function create_id($name='DEFAULT_SEQUENCE') {
        $db = self::get_connection();
        $id = $db->nextId($name);
        if (DB::isError($db))
            die($db->getDebugInfo());
        return $id;
    }
}
```

```
function rollback() {
    self::$db_handle->rollback();
    self::$db_handle = null;
}

function commit() {
    self::$db_handle->commit();
    self::$db_handle = null;
}

function close() {
    self::$db_handle->disconnect();
    self::$db_handle = null;
}
}
?>
```

Anhang B (Zugriff auf Oracle DB)

Datei: h:\schmidt\xampp\htdocs\wwwenv\or-mondial-example\mondial-access.php

```
<?
include_once('DB_Util.php');

DB_Util::connect("oci8://wi2:wi2@widb1");

$sql = "
select name, province
  from mondial.city
 where country = ?
 order by name";

$list = DB_Util::query($sql, array('F'));

foreach ($list as $city) {
    print $city['NAME']."    (".$city['PROVINCE'].")\n";
}
```

```
}  
DB_Util::close();  
?>
```

Hinweise:

bisher keine ;-)