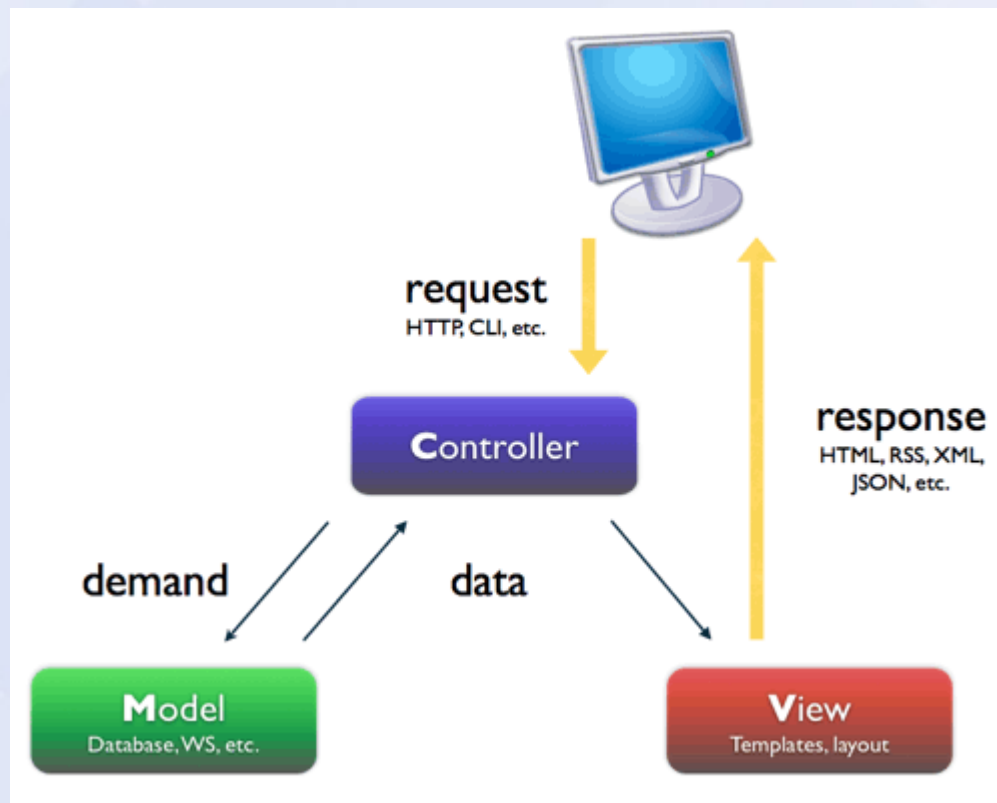


PDO – PHP Data Objects

(mgr inż. Marek Downar)

Typowa aplikacja internetowa



Baza danych

Przechowywanie danych z myślą jedynie o danych to najkrótsza droga do katastrofy.

Baza danych (1NF)

- Zapewnienie atomowości

- Stosując opis słowny tracimy przynajmniej jedną z poważnych zalet baz danych, mianowicie możliwość szybkiego wyszukiwania informacji
- Index tekstowy – brak możliwości obsługi modyfikacji w czasie rzeczywistym

Wprowadzanie danych to procedura podatna na błędy (ochrona poprawności danych) – identyfikacja klucza głównego – unikatowej charakterystyki każdego wiersza. Wybór np. nazwy klienta w charakterze identyfikatora narzuca konieczność zachowania określonego standardu zapisu, aby uniknąć wieloznaczności

Baza danych (1NF)

- Klienci powinni być identyfikowani na podstawie standardowej nazwy skróconej lub unikatowego kodu.

1NF – pierwsza postać normalna

Klucz główny powinien charakteryzować dane. Gdy wszystkie identyfikatory są atomowe i zostały zidentyfikowane klucze główne.

Baza danych (2NF)

- Sprawdzenie zależności od klucza głównego. Nadmiarowość danych, wydajność operacji

2NF – druga postać normalna

Usunięcie z tabel wszystkich atrybutów uzależnionych od części klucza.

Baza danych (3NF)

- Sprawdzenie zależności atrybutów

3NF – trzecia postać normalna

Wartości atrybutu nie daje się pozyskać na podstawie wartości innych atrybutów oprócz tych wchodzących w skład unikalnego klucza

Baza danych - normalizacja

- Prawidłowo znormalizowany model chroni przed zagrożeniami w wyniku ewolucji wymagań.
- Normalizacja minimalizuje duplikację danych

Proces normalizacji polega na zastosowaniu na szeroką skalę pojęcia atomowości w modelowanym świecie.

Baza danych - null

”Jak wiemy, są rzeczy znane. Rzeczy, o których wiemy, że o nich wiemy. Wiemy jednak również o znanych nam rzeczach nieznanach. Czyli wiemy, że istnieją rzeczy, o których nie wiemy. Są jednak także nieznanne nam rzeczy nieznanne, to znaczy te, o których nie wiemy, że o nich nie wiemy”

Transakcje

Spięcie szeregu operacji bazodanowych w w logiczną i niepodzielną całość.

A – Atomowość (ang. Atomicity)

C – Spójność (ang. Consistency)

I – Izolacja (ang. Isolation)

D – Trwałość (ang. Durability)

Konflikty - zapis-zapis

- utrata aktualizacji

Gdy transakcja T1 zapisuje wartość zmienioną przez

transakcją T2 (ignoruje zmiany przez nią dokonane).

Konflikty – zapis-odczyt

- Brudny odczyt

Gdy transakcja T2 odczytuje wartość zmienioną przez transakcję T1 przed jej zatwierdzeniem (przed wywołaniem funkcji commit). Transakcja T1 zostaje anulowana (przywrócona poprzednia wartość). Transakcja T2 odczytała niewłaściwą wartość.

Konflikty – zapis-odczyt

- Niepowtarzalny odczyt (fuzzy read)

Gdy T1 odczyta wartość, T2 ją zmieni i T1 ponownie odczyta tę samą wartość.

Konflikty - zapis-odczyt

- Fantomy (phantom read)

Gdy T1 odczytuje wartość, następnie T2 dodaje nową wartość, a T1 ponownie odczytuje zawartość bazy danych.

Poziomy izolacji

Definicja kiedy i jak zmiany dokonane przez jedną instrukcję dostępne są dla instrukcji wykonywanej równoległe.

- Read uncommitted (dirty reads)
- Read committed (non-repeatable reads)
- Repeatable read (phantom reads) dane odczytywane nie mogą być zmienione
- Serializable – brak równoczesnego wykonywania operacji na tych samych danych

Konflikty - poziomy izolacji

	Dirty read	Non-repeatable read	phantoms
Read uncommitted	V	V	V
Read committed	X	V	V
Repeatable read	X	X	V
serializable	X	X	X

Niedogodności

- Konieczność znajomości funkcji charakterystycznych dla danego silnika,
- Decyzja o zmianie silnika,
- Aplikacja wykorzystująca wiele baz danych,
- Migracja bazy danych,

PDO

Biblioteka PHP ujednocająca korzystanie z różnych baz danych. Kod tworzony przy jej pomocy staje się przenośny, niezależny od dostawcy bazy danych.

- Wsparcie dla:
 - DLIB: FreeTDS/Microsoft SQL Server/
Sybase, Firebird, IBM, MySQL, 4D, OCI,
INFORMIX, PGSQL, SQLite, ODBC

PDO – Instalacja

- Statycznie bądź dynamicznie,
- Najpierw ładowany musi być moduł pdo dopiero później moduł charakterystyczny dla danej bazy danych (np. pdo_mysql),

PDO – nawiązanie połączenia

- wykorzystanie DSN (ang. Data Source Name) – informacja o źródle danych zgodna z interfejsem ODBC (ang. Open Database Connectivity) postaci:

”rodzaj:host=nazwa;dbname=baza”

Przykład

```
$db = new  
PDO('mysql:host=153.19.55.227;port=3306;dbname=dziekanat');
```

Uwaga

W przypadku wystąpienia błędów w połączeniu, wygenerowany zostaje wyjątek PDOException, który należy obsłużyć (niebezpieczeństwo ujawnienia użytkownikowi końcowemu informacji poufnych)

PDO – połączenie

- Domyślnie:
 - Po pomyślnym nawiązaniu połączenia tworzony jest obiekt, który jest odpowiedzialny m.in. za czas trwania połączenia.
- Persistent connections:
 - Podczas nawiązywania (konstruktor!) połączenia należy ustawić flagę `PDO::ATTR_PERSISTENT`

PDO – operacje

Pobieranie danych:

```
$db → query('SELECT * FROM studenci');
```

Aktualizacja, wstawianie danych:

```
$db → exec("INSERT INTO `studenci`  
(`imie`, `nazwisko`) VALUES ('Jan','Kowalski')");
```

Usuwanie danych:

```
$db → exec("DELETE FROM studenci WHERE  
imie='Jan'");
```

PDO - transakcje

```
try{  
    $db → beginTransaction();  
    //zestaw operacji  
    $db → commit();  
}catch(Exception $e){  
    $db → rollBack();  
}
```

PDO – prepared statements

- Prepared statement – szablon zapytania SQL podany w celu jego analizy, kompilacji i optymalizacji

```
$stmt = $db → prepare("INSERT INTO student  
(imie,nazwisko) VALUES (:imie, :nazwisko)");
```

```
$stmt → bindParam(':imie',$imie_studenta);
```

```
$stmt → bindParam(':nazwisko',$nazwisko_studenta);
```

```
$imie_studenta = "Jan"; $nazwisko_studenta="Kowalski";
```

```
$stmt → execute();
```

PDO – stored procedures

- Stored procedure – procedura osadzona na serwerze bazy danych

```
$stmt = $db → prepare("CALL  
                                sp_cnt__nieobecnych(?)");  
$stmt → bindParam(1, $return_value,  
                                PDO::PARAM_INT);  
$stmt → execute();
```

PDO – obsługa wyjątków

- PDO::ERRMODE_SILENT

Domyślny tryb

(PDO::errorCode(), PDO::errorInfo())

- PDO::ERRMODE_WARNING

dodatkowo informacja w postaci ostrzeżenia (bez przerywania wykonywania aplikacji)

- PDO::ERRMODE_EXCEPTION

wygenerowanie wyjątku - PDOException

PDO - LOB

- Obsługa "dużych danych"
- Ustawienie `PDO::PARAM_LOB` pozwala na operacje na parametrze jak na strumieniu, przy wykorzystaniu takich funkcji jak `fread()`, `fgets()`
- Parametr ten ustawiany jest w funkcjach `bindParam()` oraz `bindColumn()`

```
PDO {  
    __construct ( string $dsn [, string $username [, string $password [, array $driver_options ]]] )  
    bool beginTransaction ( void )  
    bool commit ( void )  
    mixed errorCode ( void )  
    array errorInfo ( void )  
    int exec ( string $statement )  
    mixed getAttribute ( int $attribute )  
    array getAvailableDrivers ( void )  
    string lastInsertId ([ string $name = NULL ] )  
    PDOStatement prepare ( string $statement [, array $driver_options = array() ] )  
    PDOStatement query ( string $statement )  
    string quote ( string $string [, int $parameter_type = PDO::PARAM_STR ] )  
    bool rollBack ( void )  
    bool setAttribute ( int $attribute , mixed $value )  
}
```

```
PDOStatement implements Traversable {
```

```
    readonly string $queryString;
```

```
    bool bindColumn ( mixed $column , mixed &$param [, int $type [, int $maxlen [, mixed $driverdata ]]] )
```

```
    bool bindParam ( mixed $parameter , mixed &$variable [, int $data_type = PDO::PARAM_STR [, int $length [, mixed $driver_options ]]] )
```

```
    bool bindValue ( mixed $parameter , mixed $value [, int $data_type = PDO::PARAM_STR ] )
```

```
    bool closeCursor ( void )
```

```
    int columnCount ( void )
```

```
    bool debugDumpParams ( void )
```

```
    string errorCode ( void )
```

```
    array errorInfo ( void )
```

```
    bool execute ( [ array $input_parameters = array() ] )
```

```
    mixed fetch ( [ int $fetch_style = PDO::FETCH_BOTH [, int $cursor_orientation = PDO::FETCH_ORI_NEXT [, int $cursor_offset = 0 ]]] )
```

```
    array fetchAll ( [ int $fetch_style = PDO::FETCH_BOTH [, int $column_index = 0 [, array $ctor_args = array() ]]] )
```

```
    string fetchColumn ( [ int $column_number = 0 ] )
```

```
    mixed fetchObject ( [ string $class_name = "stdClass" [, array $ctor_args ] ] )
```

```
    mixed getAttribute ( int $attribute )
```

```
    array getColumnMeta ( int $column )
```

```
    bool nextRowset ( void )
```

```
    int rowCount ( void )
```

```
    bool setAttribute ( int $attribute , mixed $value )
```

```
    bool setFetchMode ( int $mode )
```

```
}
```

```
PDOException extends RuntimeException {  
    public array $errorInfo ;  
    protected string $message ;  
    protected string $code ;  
    final public string Exception::getMessage ( void )  
    final public Exception Exception::getPrevious ( void )  
    final public int Exception::getCode ( void )  
    final public string Exception::getFile ( void )  
    final public int Exception::getLine ( void )  
    final public array Exception::getTrace ( void )  
    final public string Exception::getTraceAsString ( void )  
    public string Exception::__toString ( void )  
    final private void Exception::__clone ( void )  
}
```

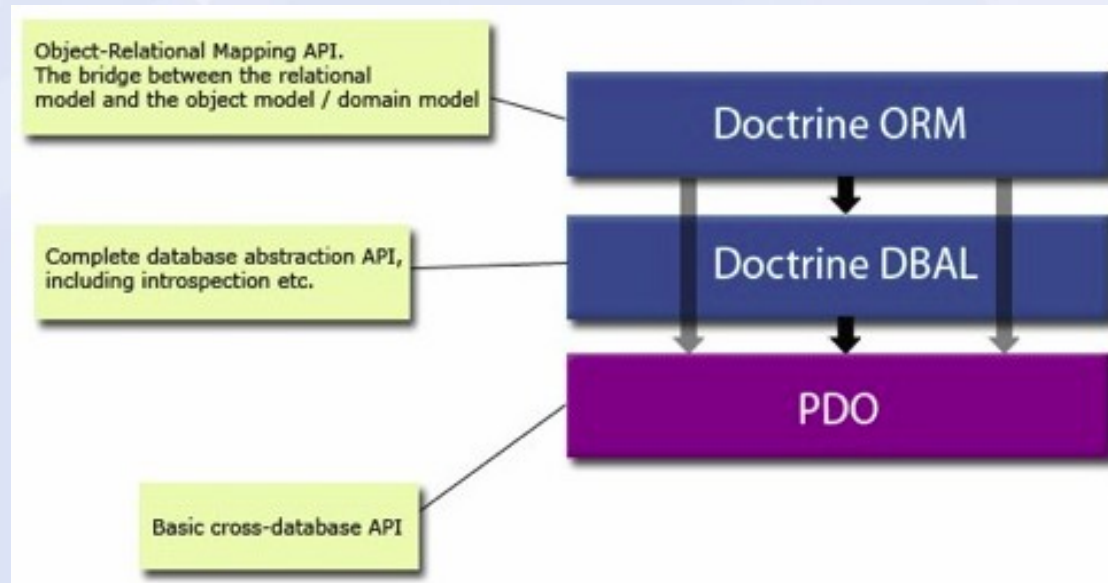
Propel

- Odwzorowanie obiektowo-relacyjne
 - Klasy PHP odwzorowywane są w tabelę bazy danych.

Przykład z strony (www.propelorm.org)

```
$book = BookPeer::retrieveByPK(123);
$book->setName('Don't be Hax0red!');
$book->save();
$criteria = new Criteria();
$criteria->add(BookPeer::PUBLISH_YEAR, 2009);
$criteria->addAscendingOrderBy(AuthorPeer::LAST_NAME);
$books = BookPeer::doSelectJoinAuthor($criteria);
foreach($books as $book) {
    echo $book->getAuthor()->getFullName();
}
```

Doctrine



- ORM dla PHP 5.2.3+ oparty na abstrakcyjnej warstwie dostępu do bazy danych (DBAL)
- Zapytania w postaci DQL wzorowanym na HQL

Doctrine przykład(1)

```
SELECT
```

```
u.id AS u__id,
```

```
u.is_active AS u__is_active,
```

```
u.is_super_admin AS u__is_super_admin,
```

```
u.first_name AS u__first_name,
```

```
u.last_name AS u__last_name,
```

```
u.username AS u__username,
```

```
u.password AS u__password,
```

```
u.type AS u__type,
```

```
u.created_at AS u__created_at,
```

```
u.updated_at AS u__updated_at,
```

```
p.id AS p__id,
```

```
p.user_id AS p__user_id,
```

```
p.phonenumber AS p__phonenumber
```

```
FROM user u
```

```
LEFT JOIN phonenumber p ON u.id = p.user_id
```

Doctrine – przykład (2)

```
$users = Doctrine_Core::getTable('User')->findAll();
foreach($users as $user) {
    echo $user->username . " has phonenumber: \n";
    foreach($user->Phonenumbers as $phonenumber) {
        echo $phonenumber->phonenumber . "\n";
    }
}
```

Doctrine – przykład (3)

```
$q = Doctrine_Query::create()
```

```
->from('User u')
```

```
->leftJoin('u.Phonenumbers p');
```

```
$users = $q->execute();
```

```
foreach($users as $user) {
```

```
    echo $user->username . " has phonenumbers: \n";
```

```
    foreach($user->Phonenumbers as $phonenum) {
```

```
        echo $phonenum->phonenum . "\n";
```

```
    }
```

```
}
```