4 JAM BERLAYAR

MENUJU PULAU DOCTRINE

Bersama

Yaya Wihardi

Irvan Riswanto

Versi 1.0

Laboratorium Basis Data
Progam Ilmu Komputer
Universitas Pendidikan Indonesia

Apa ORM itu?

ORM (Object Relational Mapping) adalah teknik pemrograman yang digunakan untuk memetakan Relasional Database ke dalam Virtual Database yang dapat diakses langsung melalui bahasa pemrograman, salah satunya dipetakan ke dalam object-object Model.

Apa Doctrine itu?

Doctrine merupakan salah satu ORM yang berjalan di PHP 5.2.x ke atas. Dikembangkan oleh doctrine-project.org. pada saat ini versi stabil terbarunya telah mencapai versi 1.2.1. Namun dalam modul ini hanya akan digunakan versi 1.1.6.

Prasyarat untuk Menggunakan Modul ini

- ✓ Menguasai PHP
- ✓ Menguasai MySQL

Persiapan Perbekalan

Installasi

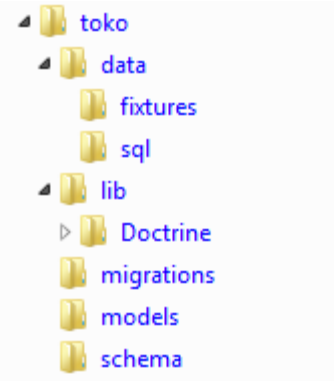
Doctrine bisa diinstal melalui berbagai cara, namun pada modul ini, digunakan doctrine sandbox yang dapat di download di : http://www.doctrine-project.org/download/1_1_6/sandbox/1/format/tgz . Pada saat modul ini dibuat, versi terbarunya yang stabil sudah mencapai versi 1.1.6.

Untuk installasi ikuti langkah-langkah berikut:

1. Pastikan Di Web Server Anda terinstall PHP 5.2.x (silahkan cari referensi lain)
2. Pastikan Versi MySQL yang anda gunakan minimal 4.x
3. Download master Doctrine di : http://www.doctrine-project.org/download/1_1_6/sandbox/1/format/tgz
4. Ekstraksi file "Doctrine-1.1.6-Sandbox.tgz" di webroot atau htdocs
5. Rename direktori hasil ekstraksi "Doctrine-1.1.6-Sandbox", menjadi direktori applikasi yang akan dibuat, dalam hal ini *"toko"*.
6. Sampai tahap ini, installasi telah berhasil dilakukan.

Struktur Direktori

Setelah melakukan installasi, di dalam folder toko, kita akan menemukan struktur direktori sebagai berikut :



Keterangan Isi Direktorinya :

File/Folder	Keterangan
./toko/data	Terdiri dari folder fixtures dan sql. fixtures digunakan untuk menyimpan file yaml yang digunakan untuk melakukan import data. sql digunakan untuk menyimpan file schema hasil export dalam format sql
./toko/lib	Di dalam direktori ini, tersimpan seluruh librari doctrine yang dibutuhkan

File/Folder	Keterangan
./toko/migrations	Tempat Menyimpan class-class migratios yang telah dibuat/digerate, dan akan digunakan dalam proses migrasi
./toko/models	Tempat Menyimpan class-class model yang telah dibuat/digerate
./toko/schema	Tempat menyimpan file Skema database dalam format YAML yang digunakan untuk menggenerate struktur table dan model
./toko/config.php	Berisi konfigurasi PATH dan konfigurasi Doctrine untuk siap pakai
./toko/doctrine	File yang dapat dieksekusi (CLI) untuk membantu bekerja dengan Doctrine
./toko/doctrine.php	File Konfigurasi untuk CLI (default dibuat oleh pengembang)
./toko/index.php	Contoh penggunaan Doctrine Sederhana

Konfigurasi Koneksi dan Database

Sebelum memulai bekerja**,** mari kita persiapkan konfigurasi database yang dibutuhkan

1. Buat database yang akan digunakan dalam applikasi, dalam hal ini diberinama : *"toko"*
2. Buka file ./toko/config.php kemudian ubah kode berikut :

```
define('DSN', 'mysql://root:@localhost/doctrine11sandbox');
```

menjadi :

```
define('DSN', 'mysql://root:@localhost/toko');
```

Catatan :

```
user     = root
password= ""
host     = localhost
database= toko
```

dan tambahkan baris berikut setelah baris terakhir :

```
Doctrine::loadModels('models');
```

3. Finish…

Perjalanan Jam Pertama

Mengangkat Jangkar dan Membentangkan Layar

Persiapan Schema

Skema/Schema, merupakan definisi database yang akan digunakan dalam sistem yang kita kembangkan. Skema ini harus disimpan di dalam folder ./toko/schema.

Untuk mempermudah dalam melakukan pemeliharaan sistem, maka skema database yang akan dibuat disimpan dan didefinisikan dalam format YAML. Adapun aturan/syntax penulisannya adalah sebagai berikut:

```
<NamaModel>:
      tableName: <nama_table>
      columns:
            <nama_field>:
                  type: <type_data>
                  primary: <true/false>
                  notnull: <true/false>
                  default: <nilai_default>
      <field_ke_n>

<ModelKe-n>
```

Catatan :

- ✓ Gunakan spasi untuk mengganti karakter TAB.
- ✓ Tambahkan spasi setelah tanda titik dua (":").
- ✓ Penamaan *identifier* hanya boleh mengandung huruf dan under score (_)

Type data yang bisa digunakan:

Type Data	Keterangan
integer(n)	Bilangan bulat, n = nilai maksimal
string(n)	Teks/Karakter, n=panjang karakter
boolean	Hanya bernilai TRUE/FALSE
timestamp	Tanggal dan Waktu dengan format : "YYYY-MM-DD H:M:S"
date	Tanggan dengan format : "YYYY-MM-DD"
time	Waktu dengan format: "H:M:S"
float	Bilangan Real
decimal(n)	Bilangan Desimal dengan n buah di belakang koma
enum	Type data yang nilainya sudah didefinisikan

Dalam modul ini, kita akan membuat schema untuk aplikasi penjualan sederhana. Silahkan buat schema berikut, kemudian simpan di folder ./toko/schema dengan nama toko_skema.yml.

<skema_toko.yml>

```
detect_relations: true
actAs: [Timestampable]
package: Master
options:
 type: MyISAM

Pelanggan:
  tableName: pelanggan
  columns:
    nama_pelanggan: string(50)

Kasir:
  tableName: kasir
  columns:
    nama_kasir: string(50)

Transaksi:
  tableName: transaksi
  columns:
    pelanggan_id: integer
    kasir_id: integer

Produk:
  tableName: produk
  columns:
    nama_produk: string(50)
    harga: float
    kategori_id: integer

Kategori:
  tableName: kategori
  columns:
    nama_kategori: string(50)

DetailTransaksi:
  tableName: detail_transaksi
  columns:
    produk_id: integer
    transaksi_id: integer
    jumlah: integer
```

Di dalam file skema di atas, kita mendefinisikan enam table dan enam model. Nama table dan nama model boleh berbeda, tapi disarankan sama. Untuk penulisan nama table disarankan menggunakan huruf capital di awal setiap kata yang mendefinisikan nama model tersebut.

Dalam definisi file YAML di atas, berikut penjelasannya :

detect_relations: true

Mendefinisikan, bahwa CLI harus melakukan deteksi dan membuat Relasi antar model secara secara otomatis.

```
actAs: [Timestampable]
```

Mendefinisikan agar di setiap class model yang dibuat dapat melakukan pencatatan tanggal pembuatan dan tanggal update untuk setiap data yang tersimpan di dalamnya.

```
package: Master
```

Mendefinisikan package yang digunakan, yaitu Master

```
options:
  type: MyISAM
```

Mendefinisikan tipe table yang akan digunakan di dalam database.

Pembuatan Model

Model dalam konsep MVC, secara sederhana diartikan sebagai representasi dari table-tabel dan relasinya yang tersimpan di dalam database.

Pembuatan model ini sendiri, dapat dilakukan secara manual, maupun secara otomatis dengan bantuan tool yang telah disediakan oleh doctrine. Untuk mempermudah proses pengembangan dan pemeliharaan, disarankan untuk menggunakan tool, karena dengan bantuan tool ini, maka model, dan database (termasuk table dan relasinya) akan secara otomatis dibuatkan/digenerate dari file skema yang tersimpan di dalam folder ./toko/schema.

Untuk membuat model secara otomatis, langkah-langkahnya sebagai berikut:

1. Buka Comand Line (cmd)
2. Kemudian pindahkan direktori kerja ke interpreter php yang dimiliki webserver.
   ```
   >Cd c:\xampp\php
   ```
3. Eksekusi file : .\toko\doctrine dengan parameter build-all-reload
   ```
   >php c:\xampp\toko\doctrine build-all-reload
   ```
4. Masukan “y”, jika muncul pertanyaan konfirmasi.
5. Finish

Sampai tahap ini, kita telah berhasil membuat model sesuai yang didefinisikan di dalam ./toko/schema/toko_skema.yml. Model yang telah dibuat bisa dilihat di ./toko/models.

Perjalanan Jam Kedua

Selamat Tinggal Pantai

Menggunakan Doctrine

Untuk menggunakan Doctrine, kita harus mereferensi file ./toko/config.php ke aplikasi yang akan kita buat. Kembali pada tujuan kita untuk membuat aplikasi toko sederhana, maka kita akan mencoba membuat formulir input untuk pelanggan.

1. Buat direktori baru dengan nama ‘aplikasi’ di folder ./toko
2. Buat file add_pelanggan.php di folder ./toko, kemudian buat kode berikut:

```
<?php
 //Menyertakan Doctrine untuk digunakan
 require_once('config.php');

 //Mengecek ketersediaan value di variable POST
 if(!empty($_POST['Pelanggan'])){
     //Membuat/Instansiasi Object Pelanggan
    $pel = new Pelanggan();
    //mengisi object pelanggan dengan data yang dikirim melalui variable POST
    $pel->fromArray($_POST['Pelanggan']);
    //menyimpan data object pelanggan ke dalam database
    $pel->save();
    $pesan = "Data Berhasil Disimpan";
 }
 include("aplikasi/view_add_pelanggan.php");
?>
```

3. Buat file view_pelanggan.php di folder ./toko/aplikasi, kemudian isi dengan kode berikut:

```
<h2>Penambahan data Pelanggan</h2>
<br><br>
<?php if(!empty($pesan)) :?>
   <?php echo $pesan ?><br>
<?php endif ?>
<form method='POST' action='add_pelanggan.php'>
   Nama Pelanggan :
   <input type='text' name='Pelanggan[nama_pelanggan]'/>
   <input type='submit' value='Simpan'/>
</form>
```

Sampai di sini kita sudah berhasil membuat aplikasi untuk penambahan data pelanggan. Silahkan ditest, dengan url : localhost/toko/add_pelanggan.php.

Untuk tahap selanjutnya, anda dapat membuat aplikasi penambahan Produk, kategori Produk, dan Kasir.

Menampilkan Data

Untuk menampilkan data, doctrine telah menyediakan magic method dan magic property untuk setiap model yang ada. Struktur bahasa yang digunakan untuk proses pengambilan data ini disebut Doctrine Query Language (DQL).

Sekarang mari kita buat aplikasi yang menampikan data pelanggan.

1. Buat file pelanggan.php di folder ./toko, kemudian buat kode berikut:

```
<?php
   require_once('config.php');

   //Mengambil data dengan query ‘Select * from pelanggan’
   $data = Doctrine_Query::create()
         ->select('*')
         ->from('Pelanggan')
         ->execute();

   include("aplikasi/view_pelanggan.php");
?>
```

2. Buat file view_pelanggan.php di folder ./toko/aplikasi, kemudian isi dengan kode berikut:

```
<h2>Daftar Pelanggan</h2>
<br><br>
<?php no=1 ?>
<table border=1px>
   <thead>
      <tr>
         <th width='20px'>No.</th>
         <th>Nama Pelanggan</th>
         <th>Tanggal Pendaftaran</th>
         <th>Aksi</th>
      </tr>
   </thead>
   <tbody>
      <?php foreach($data as $detail): ?>
         <tr>
            <td><?php echo $no++ ?></td>
            <td><?php echo $detail->nama_pelanggan ?></td>
            <td><?php echo $detail->created_at ?></td>
```

```
          <td>
            <a href="edit_pelanggan.php?id=<?php echo $detail->id ?>">ubah</a> |
            <a href="delete_pelanggan.php?id=<?php echo $detail->id ?>">hapus</a>
          </td>
        </tr>
      <?php endforeach ?>
    </tbody>
</table>
```

Sampai di sini kita sudah berhasil membuat aplikasi untuk menampilkan data pelanggan. Silahkan ditest, dengan url : localhost/toko/pelanggan.php.

Melakukan Updating Data

Sekarang kita akan membuat aplikasi untuk melakukan pengubahan/updating data.

1. Buat file edit_pelanggan.php di folder ./toko, kemudian buat kode berikut:

```
<?php
  require_once('config.php');

  if(!empty($_POST['Pelanggan'])){
    //Mencari data yang akan diupdate
    $pel = Doctrine::getTable('Pelanggan')->find($_POST['Pelanggan']['id']);
    //Mengisi data yang telah diubah untuk object Pelanggan dengan data yang dipost,
    $pel->fromArray($_POST['Pelanggan']);
    //Menyimpan data yang telah diperbaharui
    $pel->save();

    $pesan = "Data Berhasil Disimpan";
  }else{
    //MEncari data berdasarkan id yang diketahui
    $pelanggan = Doctrine::getTable('Pelanggan')->find($_REQUEST['id']);
  }
  include("aplikasi/view_edit_pelanggan.php");?>
```

2. Buat file view_edit_pelanggan.php di folder ./toko/aplikasi, kemudian isi dengan kode berikut:

```
<h2>Pemutakhiran data Pelanggan</h2>
<br><br>
<?php if(!empty($pesan)) :?>
  <?php echo $pesan ?><br>
<?php endif ?>
<form method='POST' action='edit_pelanggan.php'>
```

```
  Nama Pelanggan :
  <input type='text' name='Pelanggan[nama_pelanggan]'
    value="<?php echo $pelanggan->nama_pelanggan?>"/>
  <input type='hidden' name='Pelanggan[id]' value="<?php echo $pelanggan->id ?>"/>
  <input type='submit' value='Ubah'/>
</form>
```

Sampai di sini kita sudah berhasil membuat aplikasi untuk pengubahan data pelanggan. Silahkan ditest, dengan url : localhost/toko/pelanggan.php, kemudian pilih link ubah di kolom aksi.

Melakukan Penghapusan Data

Tidak lengkap rasanya jika aplikasi penambahan tidak dilengkapi dengan penghapusan. So, mari kita mulai membuat aplikasi penghapusannya.

1. Buat file delete_pelanggan.php di folder ./toko, kemudian buat kode berikut:

```
<?php
  require_once('config.php');

  if(!empty($_REQUEST['id'])){
    //Mencari data yang akan dihapus
    $pel = Doctrine::getTable('Pelanggan')->find($_REQUEST['id']);
    //Menghapus data yang dimaksud
    $pel->delete();

    $pesan = "Data Berhasil Disimpan";
  }else{
    $pesan = “Maaf ID Pelanggan kosong, silahkan cek ulang”;
  }
  include("aplikasi/view_edit_pelanggan.php");?>
```

2. Buat file view_edit_pelanggan.php di folder ./toko/aplikasi, kemudian isi dengan kode berikut:

```
<?php if(!empty($pesan)) :?>
  <h2><?php echo $pesan ?></h2>
<?php endif ?>
```

Sampai di sini kita sudah berhasil membuat aplikasi untuk pengubahan data pelanggan. Silahkan ditest, dengan url : localhost/toko/pelanggan.php, kemudian pilih link hapus di kolom aksi.

Ok, sampai saat ini, aplikasi pengelolaan pelanggan sudah selesai kita buat. Sekarang, silahkan integrasikan dengan membuat link di setiap halaman yang dibutuhkan. Dan anda dapat membuat aplikasi lainnya untuk pengelolaan PRoduk, Kategori PRoduk, dan Kasir.

!! Coba Ingat Lagi Tentang

- ✓ **$pel = new Pelanggan()**
- ✓ **$pel = Doctrine::getTable('Pelanggan')->find($_REQUEST['id'])**
- ✓ **$pel->fromArray($_POST['Pelanggan'])**
- ✓ **$pel->save()**
- ✓ **$pel->delete()**
- ✓ **$data = Doctrine_Query::create()**
  ->select('*')
  ->from('Pelanggan')
  ->execute()

Perjalanan Jam Ketiga

Menerjang Badai Query dan Agregasi

Searching

Dalam SQL, proses pencarian dipastikan menggunakan syntax where. Begitupun dengan doctrine, kita dapat menggunakan method where di setiap model yang telah dibuat.

Ok, mari kita coba mencicipinya dengan membuat aplikasi filter produk berdasarkan kategori.

1. Buat file produk.php di folder ./toko, kemudian buat kode berikut:

```
<?php
    require_once('config.php');
    //Mengambil data Kategori untuk dijadikan pilihan combobox
    $data = Doctrine_Query::create()
              ->select('*')
             ->from('Kategori')
             ->execute();

    //Mengecek keberadaan nilai kategori_id untuk melakukan filtering data produk
    if(!empty($_POST['kategori_id'])){
             //Mengambil data produk dengan filter kategori_id
             $cari = Doctrine_Query::create()
                       ->select('*')
                       ->from('Produk p')
                       ->leftJoin('Kategori k')
                       ->where("p.kategori_id=".$_POST['kategori_id'])
                       ->execute();
    }
    include("aplikasi/view_produk.php");
?>
```

2. Buat file view_produk.php di folder ./toko/aplikasi, kemudian isi dengan kode berikut:

```
<h2>Daftar Produk Berdasarkan Kategori</h2>
<br><br>
<form method='POST' action='cari.php'>
    Kategori :
    <select name='kategori_id'>
      <?php foreach($data as $detail): ?>
        <option value=<?php echo $detail->id ?>><?php echo $detail->nama_kategori ?></option>
      <?php endforeach ?>
    </select>
```

```
    <input type='submit' value='Cari'/>
</form>

<?php if(!empty($cari)){?>
<?php $no=1 ?>
<table border=1px>
  <thead>
    <tr>
      <th width='20px'>No.</th>
      <th>Nama Produk</th>
      <th>harga</th>
    </tr>
  </thead>
  <tbody>
    <?php foreach($cari as $detail): ?>
      <tr>
        <td><?php echo $no++ ?></td>
        <td><?php echo $detail->nama_produk ?></td>
        <td><?php echo $detail->harga ?></td>
      </tr>
    <?php endforeach ?>
  </tbody>
</table>
<?php } else { ?>
    Silahkan pilih kategori terlebih dahulu
<?php }?>
```

Ok, mari kita coba dengan url : localhost/toko/produk.php, kemudian pilih kategori yang anda inginkan.

Sejauh ini kita telah mengenal method leftJoin() dan where(). Method where digunakan untuk melakukan filtering seperti halnya di SQL biasa. Sedangkan leftJoin hanya dibutuhkan jika kita akan menggunakan field dari table lain yang menjadi referensi table yang deselect. Selebihnya tidak, karena Join dapat dihandle secara otomatis oleh Doctrine_Query.

Magic Join

Magic join, merupakan salah satu fitur yang diberikan oleh doctrine. Prinsipnya, semua table yang berelasi, akan secara otomatis dijoin oleh doctrine tanpa mendefinisikan join/leftJoin di bagian DQL yang kita buat. Baik kali ini kita coba untuk membuat daftar penjualan produk dan pembelinya untuk hari ini.

1. Buat file harian.php di folder ./toko, kemudian buat kode berikut:

```
<?php
    require_once('config.php');

    $data = Doctrine_Query::create()
            ->select('*')
            ->from('DetailTransaksi dt')
            ->where('date(dt.created_at) = date(now())')
            ->execute();

    include("aplikasi/view_harian.php");
?>
```

2. Buat file view_harian.php di folder ./toko/aplikasi, kemudian isi dengan kode berikut:

```
<h2>Laporan Penjualan Harian</h2>
<br><br>

<?php $no=1 ?>
<table border=1px>
  <thead>
    <tr>
      <th width='20px'>No.</th>
      <th>Nama Produk</th>
      <th>Kategori Produk</th>
      <th>Pembeli</th>
      <th>Kasir yang melayani</th>
    </tr>
  </thead>
  <tbody>
    <?php foreach($cari as $detail): ?>
      <tr>
        <td><?php echo $no++ ?></td>
        <td><?php echo $detail->Produk->nama_produk ?></td>
        <td><?php echo $detail->Produk->Kategori->nama_kategori ?></td>
        <td><?php echo $detail->Transaksi->Pelanggan->nama_pelanggan ?></td>
        <td><?php echo $detail->Transaksi->Kasir->nama_kasir ?></td>
      </tr>
    <?php endforeach ?>
  </tbody>
</table>
```

Mari kita lihat contoh kode di atas. Kita melakukan select dari DetailTransaksi, tapi karena Detail Transaksi berelasi dengan Transaksi dan Transaksi Sendiri berelasi dengan Pelanggan dan KAsir, maka tanpa melakukan join sekalipun kita dapat mengambil nama pelanggan dengan kode :

```
$detail->Transaksi->Pelanggan->nama_pelanggan
```

Dan mengambil data kasir dengan :

```
$detail->Transaksi->Kasir->nama_kasir
```

Disamping itu, DetailTransaksi berelasi dengan PRoduk, dan Produk sendiri berelasi dengan Kategori, maka kita bisa mendapatkan data nama Produk cukup dengan :

```
$detail->Produk->nama_produk
```

Dan mendapatkan data kategori dengan :

```
$detail->Produk->Kategori->nama_kategori
```

Semua itu terjadi karena di masing-masing table telah dibuat relasi ke table-tabel reference-nya, yaitu dengan menambahkan field “transaksi_id” dan “produk_id” di table detail_transaksi, begitupun di table-tabel lainnya. Dengan penambahan field <nama_Table>_id, ini mendefinisikan relasi dari table tersebut ke table <nama_Table>. Sehingga Doctrine secara otomatis akan menerapkan magic join ke table-table yang berelasi tersebut.

Agregasi

Tak lengkap rasanya jika berbicara query tanpa membahas fungsi-fungsi agregasi yang disediakan RDBMS. Dalam sesi kali ini kita akan mencoba membahas menggunakan fungsi agregasi di dalam DQL yang kita buat.

1. Buat file laporan.php di folder ./toko, kemudian buat kode berikut:

```
<?php
    require_once('config.php');
    //MEngambil data transaksi yang digroup berdasarkan produk
    $data = Doctrine_Query::create()
            ->select('dt.*, p.*, sum(dt.jumlah) as total')
            ->from('DetailTransaksi dt')
            ->leftJoin('dt.Produk p')
            ->groupBy('p.id')
             ->execute();

    include("aplikasi/view_laporan.php");
?>
```

3. Buat file view_laporan.php di folder ./toko/aplikasi, kemudian isi dengan kode berikut:

```
<h2>Laporan Penjualan Per Produk</h2>
<br><br>
<?php $no=1 ?>
<table border=1px>
```

```
  <thead>
    <tr>
      <th width='20px'>No.</th>
      <th>Produk</th>
      <th>Kategori</th>
      <th>Jumlah Terjual (unit)</th>
    </tr>
  </thead>
  <tbody>
    <?php foreach($data as $detail): ?>
      <tr>
        <td><?php echo $no++ ?></td>
        <td><?php echo $detail->Produk->nama_produk ?></td>
        <td><?php echo $detail->Produk->Kategori->nama_kategori ?></td>
        <td><?php echo $detail->total ?></td>
      </tr>
    <?php endforeach ?>
  </tbody>
</table>
```

Nah, silahkan coba dengan url: localhost/toko/laporan.php.

Dari DQL di atas, kita dapat melihat penggunaan penggunaan fungsi SUM di bagian select. Tidak ada perbedaan dengan query biasa bukan? Begitupun dengan fungsi-fungsi agregasi lainnya. Silahkan dicoba.

Perjalanan Jam Keempat

Tiba di Pantai Native Query

Dari awal perjalanan, kita disuguhkan dengan sesuatu yang baru. Dimulai dari YMLsampai dengan DQL. Nah sekarang bagaimana jika anda tidak ingin meninggalkan bahasa kesayangan anda SQL, namun anda harus bekerja bersama-sama orang yang menyukai kemudahan Doctrine. Solusinya adalah, anda dapat menggunakan native Query.

Kelas yang dapat digunakan adalah Doctrine_Manager. Untuk itu mari kita lihat contoh penggunaanya.

1. Buatlah file native.php di ./toko, kemudian isi dengan kode berikut:

```
<?php
    $conn = Doctrine_Manager::connection();
    $query = “Select * from pelanggan”
    $stt = $conn->prepare($query);
    $stt->execute();
    $data = $stt->fetchAll() ;
    Include(“aplikasi/view_native.php”);
?>
```

2. Buatlah file view_native.php di ./toko/aplikasi. Kemudian masukan kode berikut.

```
<h2>Daftar Pelanggan</h2>
<br><br>
<?php $no=1 ?>
<table border=1px>
  <thead>
    <tr>
      <th width='20px'>No.</th>
      <th>Nama Pelanggan</th>
      <th>Tanggal Pendaftaran</th>
    </tr>
  </thead>
  <tbody>
    <?php foreach($data as $detail): ?>
      <tr>
        <td><?php echo $no++ ?></td>
        <td><?php echo $detail['nama_pelanggan'] ?></td>
        <td><?php echo $detail['created_at'] ?></td>
      </tr>
    <?php endforeach ?>
  </tbody>
</table>
```

Silahkan coba dengan mngunjungi url : localhost/toko/native.php

Terdapat beberapa catatan untuk penggunaan native query ini, yaitu:

1. Untuk melakukan select/join, yang digunakan adalah nama table di database, bukan nama model yang telah dibuat. Berbeda dengan DQL bukan.
2. Hasil dari fetching atau dari query berupa Array, bukan object seperti pada DQL. Pada contoh DQLsebelumnya, kita menggunakan $detail->nama_pelanggan untuk mendapatkan nama pelanggan, sedangkan pada contoh native query, kita menggunakan $detail['nama_pelanggan'] untuk mendapatkan nama pelanggan tsb.
3. Jika ingin melihat hasil query/fetch, coba anda gunakan perintah print_r() yang telah disediakan oleh php. Missal untuk contoh di atas, data hail fetch disimpan di variable $data, maka untuk melihat strukturnya bisa menggunakan :

```
<pre>
        <?php print_r($data) ?>
</pre>
```

Baiklah untuk native query tidak terlalu banyak yang dapat digali, tinggal berlatih lebih banyak lagi, karena yang terpenting adalah kemampuan bahasa SQL itu sendiri.

Berlabuh di Pulau Doctrine

Nah, sampai di sini, perjalanan kita telah tiba di Pulau Doctrine. Selanjutnya tugas Anda untuk melakukan penjelajahan lebih dalam lagi hingga ke pelosok-pelosok.

Untuk informasi dan referensi lebih lanjut, dapat mengunjungi: http://www.doctrine-project.org/documentation

Sampai jumpa di Ekspedisi berikutnya.

- Bravo -