

# **OOP PHP**

**Object-Oriented Programming**



**Tinju Cepat OOP dengan PHP**

**The Initial Step to Become a Greatest  
PROGRAMMER**

**GUN GUN FEBRIANZA**

This is My Contribution for Open Library Concept.

First Published 21 December 2015,

Tinju Cepat OOP dengan PHP version 1.0

*Code Example* : PHP

Download full source code : <https://github.com/PUSRISTEK>

Kritik dan saran silahkan kirim ke : [pusatrisetteknologi@gmail.com](mailto:pusatrisetteknologi@gmail.com)

Kritik dan saran terbaik dari pembaca akan saya tampilkan pada edisi revisi.

Kritik dan saran diperlukan agar ebook ini menjadi lebih baik lagi & terus berkembang.

## **Special Thanks for..**

Segala puji bagi Tuhan yang telah memudahkan saya untuk membuat buku ini agar bisa dibaca oleh orang-orang yang membutuhkan dan mempunyai tekad kuat untuk menuntut ilmu.

Thanks so much untuk Maudy Ayunda yang telah  
memotivasi dan menginspirasi saya untuk terus belajar dan berbagi  
agar kehidupan saya dan orang lain lebih baik lagi

“Kini saya sadar dan punya alasan besar untuk terus menulis dan banyak membaca sampai akhir hayat, ada manfaat besar yang tak terhitung, sulit didefinisikan dan sulit diekspresikan. Hanya bisa dirasakan dan dibayangkan. it's like loving someone.” - #GunGunFebrianza

Best Moment - 17/12/2015

## Sekilas tentang KRUNA

KRUNA adalah singkatan dari (*Knowledge Representation Using Network Algorithm*) ini adalah sebuah metodologi belajar yang saya ciptakan agar performa belajar kita menjadi lebih optimum.

Manfaatnya kita bisa memahami suatu cabang keilmuan secara terstruktur dalam artian jika kita membaca suatu buku kita bisa hafal dan faham setiap jengkalnya.

Segala informasi yang kita pelajari dan kita anggap penting akan masuk kedalam *node node* yang terstruktur membentuk akar, manfaatnya adalah jika kita lupa segala hal yang telah kita pelajari kita cukup melacaknya melalui *node node* ini.

Manfaat besarnya kita bisa mengetahui sejauh mana kapasitas kita dan potensi kita melalui *node node* ini, begitu juga progress perkembangan belajar kita. Terkadang kita perlu cermin untuk mengetahuinya.

Setelah melakukan beberapa kali penelitian metode belajar saya membuat kesimpulan Ada 5 Langkah dasar yang harus kita lakukan agar kita bisa faham apa yang ingin kita pelajari.

1. Baca
2. Fahami
3. Hafal
4. Praktek
5. Pastikan Hafal dan Faham

Tapi sebelum mengeksekusi semua hal yang ada diatas masih ada yang harus diketahui yaitu seberapa jauh Metacognition kita untuk belajar. Metacognition adalah soal seberapa besar kepekaan anda menganggap suatu hal adalah sesuatu yang sangat penting.

Jika itu benar benar sangat penting kita akan mengingatnya (tersimpan dalam long term memory) tanpa harus menghafalnya dan timbul rasa motivasi yang besar untuk belajar.

Studi Kasus :

Sering kali **Deep Metacognition** lahir dari orang orang yang memiliki latar kehidupan yang sulit, ketika mereka belajar mereka memahami setiap jengkalnya karena menganggap bahwa satu bait yang mereka baca ini sesuatu yg sangat penting dan akan sangat berpengaruh kedepanya jika tidak sempat difahami.

**Deep Metacognition** akan membuat kita terbayang tentang efek domino, jika saya tidak faham ini maka saya tidak akan faham itu? jika saya tidak faham itu saya tidak akan faham hal hal luar biasa ini? jika saya tidak faham ini hidup saya gagal? jika hidup saya gagal maka ... dst

Akan ada aplikasi khusus yang akan saya buat untuk mendukung metode belajar ini. Sebab Metode Belajar ini memang belum selesai sepenuhnya semoga diversi ebook selanjutnya saya berhasil menyelesaikanya agar bisa digunakan. Hanya secercah inspirasi.

Semoga Tulisan Singkat Ini Bermanfaat.  
Still on progress developing KRUNA. be patient.

## **Content Table**

Chapter 1 : Apa itu Object Oriented Programming?

Chapter 2 : Apa itu Class? Method dan Variable?

Chapter 3 : Membuat Sebuah Object Menggunakan Class

Chapter 4 : Memanggil Method dalam sebuah Object

Chapter 5 : Apa itu Constructor Function?

Chapter 6 : Apa itu Inheritance?

Chapter 7 : Apa itu Function Overriding?

Chapter 8 : Apa itu Access Modifier?

Chapter 9 : Apa itu PDO (PHP Data Object)?

Chapter 10 : Method Query()

Chapter 11 : Method Exec()

Chapter 12 : Try/Catch Statement

Chapter 13 : Memahami Array

Chapter 14 : Method Fetch()

Chapter 15 : Prepared Statement

Chapter 16 : Method bindParam()

Chapter 17 : Fetch Style

Chapter 18 : Finishing Touch

Special Chapter : Quadruple Attack AJAX, PHP dan MySQL

## ***Chapter 1 : Apa itu Object Oriented Programming?***

Sebelum kita memahami apa itu *Object Oriented Programming*, ada analogi yang harus kita ketahui tentang *object*.

Apa itu *object*? Kita ambil saja contoh dalam kehidupan kita sehari-hari di alam semesta ini terdapat matahari, bulan, bumi dan planet-planet lainnya. Matahari, bulan dan bumi ini disebut sebagai sebuah *object* dari sistem alam semesta.

Kemudian *object* apa saja yang ada didalam sistem komputer? pada *hardware* ada *object monitor*, *object keyboard*, *object mouse*, dan sebagainya. Begitu juga dalam perspektif OOP (*Object Oriented Programming*) segala sesuatu yang ada didunia ini bisa disebut sebagai sebuah *object* yang nantinya akan digunakan kedalam sebuah konsep untuk membuat suatu program.

Jadi *object* adalah representasi hasil pemikiran yang diterapkan untuk membuat sebuah program.

<b>Table KRUNA 1</b>	
Apa anda sudah membacanya <i>Chapter 1</i> secara terstruktur?	Ya / Tidak
Apa anda sudah faham apa itu <i>object</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Tutup halaman Apa anda sudah hafal apa itu <i>object</i> ?	Ya / Tidak

## Chapter 2 : Apa itu *Class*? *Method*? dan *Variable*?

*Class* atau kelas adalah sebuah tipe *data* yang telah diprogram. Sebuah kelas bisa memiliki sebuah *Variable* dan *Method* didalamnya. Dalam pemograman kita bisa menggunakan *Class* sebagai sebuah *Code* yang bisa kita gunakan berkali kali untuk membuat sebuah *object*. (if you forget what is 'object' read chap 1 )

Sebagai Contoh kita akan membuat sebuah *Class* bernama buku menggunakan bahasa pemograman php :

```
<?php
Class Buku{

}
?>
```

*Class* telah dibuat, selanjutnya kita bisa mengisi suatu *Variable* didalamnya yang akan kita gunakan untuk membuat sebuah *Method*. *Variable* yang akan kita buat adalah judul dan harga.

Ada yang harus dicatat tentang *Variable* dan *Method*

‘*Variable* didalam sebuah *Class*’ secara teknisnya dalam *Coding* disebut *Property*

‘*Method* didalam sebuah *Class*’ secara teknisnya dalam *Coding* disebut *Function*

Contoh *Source Code* :

```
<?php
Class Buku{

/* Variabel yang ada didalam kelas buku */
var $judul;
var $harga;

}
?>
```

Setelah membuat *Variable* kemudian kita akan membuat sebuah *Method* untuk *Class* buku, *Method* yang akan kita buat mengikuti konvensi OOP yang digunakan dibanyak bahasa pemograman (seperti *Java* dan *Ruby*) atau lebih dikenal dengan sebutan *Getter Setter Method/Function*.

*Method* yang akan kita buat sangat sederhana yaitu *Method* SetJudul, GetJudul, SetHarga dan GetHarga. *Method* SetJudul dan SetHarga digunakan untuk mengisi sebuah nilai kedalam *variable* judul dan harga sedangkan *method* GetJudul dan GetHarga digunakan untuk menampilkan nilai *variable* judul dan harga yang telah kita masukan.



Contoh *Source Code* untuk membuat *Variable* dan *Method* dalam *Class* Buku :

```
<?php
Class Buku{
/* Variabel yang ada didalam kelas buku */
var $judul;
var $harga;

/* Sekumpulan Method yang ada didalam kelas buku */
Function setHarga($parameter){
    $this->harga = $parameter;
}

/* syntax echo digunakan untuk menampilkan sebuah nilai pada output */
Function getHarga(){
    echo $this->harga . "<br/>";
}

Function setJudul($parameter){
    $this->judul = $parameter;
}

/*br adalah tag html untuk membuat garis baru */
Function getJudul(){
    echo $this->judul . " <br/>";
}
}
?>
```

\$this adalah *superVariable* yang ada didalam bahasa php, \$this mengacu pada *object* yang sama. Kita akan memahaminya lebih lanjut dibab 3.

Table KRUNA 2	
Apa anda sudah membacanya <i>Chapter 2</i> secara terstruktur?	Ya / Tidak
Apa anda sudah faham apa itu <i>Class</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda sudah faham apa itu <i>Method</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda sudah faham apa itu <i>Variable</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Tutup Halaman cek kembali Apa anda sudah hafal apa itu <i>object</i> , <i>Class</i> , <i>Method</i> dan <i>Variable</i> ?	Ya / Tidak

### ***Chapter 3 : Membuat sebuah Object menggunakan Class***

Setelah kita membuat sebuah *Class* selanjutnya kita bisa membuat sebuah *object*. *Object* ini bisa kita buat sebanyak mungkin dan memiliki *data* struktur yang sama sesuai dengan kelasnya. *Object* ini akan memiliki *Variable* dan *Method* yang sama seperti yang ada didalam *Class*nya.

Dibawah ini adalah contoh *Source Code* membuat sebuah *object* :

```
$Fisika = new Buku;  
$Matematika = new Buku;
```

**New** adalah keyword yang digunakan untuk membuat sebuah *object* dari kelas yang dimaksud. Pada *Code* diatas kita telah membuat dua *object* dan *object(s)* ini bersifat independen, selanjutnya kita bisa mengakses *Method* dan *Variable* yang ada didalamnya atau biasa disebut ***Calling Member Function***.

## Chapter 4 : Memanggil Method dalam sebuah Object

Untuk memanggil *Method* pada *object* yang telah kita buat sangat mudah. Dibawah ini adalah contoh *Source Code* memanggil *Method* pada sebuah *object* :

```
$Fisika->setJudul( "Fisika Dasar untuk Engineer" );
$Matematika->setJudul( "Matematika Dasar untuk Engineer" );

$Fisika->setHarga( 80000 );
$Matematika->setHarga( 100000 );
```

Pada *Code* diatas *object* fisika mencoba mengakses *method* *setJudul* dan *setHarga* yang ada didalam dirinya sendiri untuk mengisi *Variable* judul dan harga pada dirinya sendiri atau *object* itu sendiri begitu juga dengan *object* matematika. Untuk menampilkan *data* yang telah diisi kedalam sebuah *object* kita bisa menggunakan *Method* *get* yang telah kita buat.

```
$Fisika->getJudul();
$Fisika->getHarga();
$Matematika->getJudul();
$Matematika->getHarga();
```

Contoh Full *Code* ada pada *file* zip yang telah didownload. Nama *File* :  
OOP1\_ClassAndObject.php

Ketika dieksekusi hasilnya :



Selain mempelajari bagaimana cara *Coding* oop yang benar kita juga harus tau cara mempelajari oop yang salah. **Ini sangat berguna jika kita melakukan sebuah kesalahan *Coding*, jika kita telah mengenalinya kita bisa melacak kesalahan kita dengan cepat.**

Studi Kasus Belajar Salah
Bagaimana jika anda salah memanggil nama sebuah kelas untuk membuat sebuah <i>object</i> ? (misalkan kesalahan typo)
Bagaimana jika pada <i>object</i> fisika kelas buku anda memanggil <i>Method</i> yang tidak ada? (misalkan kesalahan typo)
Bagaimana jika anda menggunakan <i>Method</i> <i>get/set</i> pada <i>object</i> matematika dari kelas buku (Calling Member Function) tanpa memasukan terlebih dahulu keyword <i>new</i> sebelumnya untuk membuat sebuah <i>object</i> ?

## Chapter 5 : Apa itu Constructor Function?

*Constructor Function* adalah fungsi yang memiliki tipe spesial karena dia akan dipanggil secara otomatis ketika *object* dari suatu kelas dibuat. Sifat fungsi yang satu ini bisa kita manfaatkan dalam pemograman. PHP menyediakan fungsi spesial `__Construct()` untuk membuat sebuah *Constructor*. Kita bisa menambahkan banyak *argument* kedalam fungsi *constructor*.

Dibawah ini adalah contoh untuk membuat sebuah fungsi *constructor* untuk kelas buku :

```
<?php
Class Buku{
/* Variabel yang ada didalam kelas buku */
var $judul;
var $harga;

/*br adalah tag html untuk membuat garis baru */
Function GetHarga(){
echo $this->harga . "<br/>";
}

Function GetJudul(){
echo $this->judul . " <br/>";
}

/* Fungsi Spesial Construct didalam kelas buku */
Function __construct( $parameter1, $parameter2 ){
$this->judul = $parameter1;
$this->harga = $parameter2;
}
}

?>
```

Setelah kita membuat fungsi *construct* didalam *Class* buku, *Method* `setHarga` dan `setJudul` sudah tidak diperlukan lagi. Kita bisa langsung mengisi nilai untuk *Variable* `judul` dan `harga` saat *object* pertama kali kita buat. Dibawah ini adalah contohnya :

```
$Fisika = new Buku( "Fisika Dasar untuk Engineer", 85000 );
$Matematika = new Buku( "Matematika Dasar untuk Engineer", 95000 );
```

Saat kita membuat *object* Fisika kita bisa langsung mengisi sebuah nilai. `$parameter1` didalam fungsi *construct* akan membaca *parameter* pertama pada *object* fisika (“Fisika Dasar untuk Engineer”) dan menyimpannya kedalam *Variable* `judul` dan `$parameter2` didalam fungsi *construct* akan membaca *parameter* kedua pada *object* fisika (85000) dan menyimpannya kedalam *Variable* `harga`.

Setelah kita menyimpannya kita bisa memanggilnya dengan cara seperti sebelumnya. Dibawah ini contoh *Source Code* untuk memanggilnya :

```
$Fisika->getJudul();
$Fisika->getHarga();
$Matematika->getJudul();
$Matematika->getHarga();
```

Sample Full *Code* ada pada *file* zip yang telah didownload. Nama *File* :  
OOP2\_ConstructFunction.php

Ketika dieksekusi hasilnya :



Table KRUNA 3	
Apa anda sudah membacanya <i>Chapter</i> 3, 4 dan 5 secara terstruktur?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>object</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>Class</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>Method</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>Variable</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda sudah faham bagaimana cara membuat sebuah <i>object</i> menggunakan kelas yang anda buat?	Ya / Tidak
Apa anda sudah faham bagaimana cara memanggil <i>Method</i> dalam sebuah <i>object</i> yang anda buat?	Ya / Tidak
Apa anda sudah faham bagaimana cara menggunakan fungsi <i>construct</i> ?	Ya / Tidak
Apa anda sudah melakukan studi kasus belajar salah untuk fungsi <i>Construct</i> ?	Ya / Tidak
Tutup Halaman cek kembali Apa anda sudah hafal apa itu <i>construction Function</i> ?	Ya / Tidak

## Chapter 6 : Apa itu *Inheritance*?

Sebelum kita membahas apa itu *Inheritance* ada yang harus kita fahami terlebih dahulu tentang sebuah *Class*, sebelumnya kita telah membuat *Class* bernama buku. Kemudian kita ingin membuat sebuah *Class* baru bernama Komik dan pada kelas Komik kita ingin semua *Variable* dan *Method* yang ada didalam *Class* Buku ada didalam kelas Komik. Maka dalam *Object Oriented Programming* kita tidak perlu menulis ulang *Code Variable* dan *Method* yang ada didalam *Class* buku kedalam *Class* Komik.

Cukup dengan membuat *Code* seperti yang ada dibawah ini :

```
Class Komik extends Buku{  
    var $penerbit;  
  
    Function setPenerbit($parameter){  
        $this->penerbit = $parameter;  
    }  
  
    Function getPenerbit(){  
        echo $this->penerbit. "<br />";  
    }  
}
```

Pada *Code* diatas *Class* Komik secara otomatis memiliki *variable* dan *method* yang ada didalam *Class* Buku. Selanjutnya pada *Class* Komik kita bisa membuat *variable* dan *method* yang baru yaitu *variable* penerbit dan *method* set/get penerbit. Pada kasus diatas ini artinya *Class* Buku menjadi **Parent Class** dan Komik menjadi **Child Class**. Proses ini disebut dengan *Inheritance*.

**Keyword *extends*** pada *Class* Komik artinya *class* yang kita buat ini bisa mengkloning isi *code* kelas lainnya tanpa harus menulis ulang. Inilah yang menjadi keunggulan OOP tehnik ini disebut dengan ***code re-usable***.

Jadi *inheritance* secara teoritisnya adalah penurunan sebuah sifat dan secara teknisnya dalam pemograman adalah penurunan struktur *data* agar *Class* turunanya memiliki *variable* dan *method* yang sama.

Agar lebih faham kita akan memasukan *fungsi construct* kedalam *Class* Komik. (*if you forget what is construct function go back to chap 5*) Contoh *source code* ada dibawah ini:

```

Class Komik extends Buku{
    var $penerbit;

    Function setPenerbit($parameter){
        $this->penerbit = $parameter;
    }

    Function getPenerbit(){
        echo $this->penerbit. "<br />";
    }

    /* Fungsi Spesial Construct didalam kelas Komik */
    function __construct( $parameter1, $parameter2, $parameter3 ){
        $this->judul = $parameter1;
        $this->harga = $parameter2;
        $this->penerbit = $parameter3;
    }
}

```

Pada *Class* komik kita tidak melihat ada *variable* judul dan harga tapi kita bisa mengisinya melalui fungsi *construct* karena *Class* Komik mempunyai struktur *data* turunan dari *Class* Buku.

Untuk menggunakannya kita bisa membuat sebuah *object* menggunakan *Class* Komik (*if you forget how to create object from specific class go to chap 3*). *Code* dibawah ini artinya kita membuat sebuah *object* bernama **komikjenaka** menggunakan *Class* komik dan langsung mengisi 3 buah nilai kedalam *object* tersebut :

```
$komikjenaka = new Komik( "Komik si juki", 85000, "juki" );
```

Untuk memanggil *data* penerbitnya *method* yang ada didalam *object* **komikjenaka** itu sendiri bisa kita panggil dengan *code* dibawah ini :

```
$komikjenaka->getPenerbit();
```

Jika ingin memanggil *data* judul dan harga pada *object* **komikjenaka** kita bisa memanggilnya menggunakan *code* dibawah ini :

```

$komikjenaka->getJudul();
$komikjenaka->getHarga();
$komikjenaka->getPenerbit();

```

Keseluruhan *Code* :

```
<?php
class Buku{
/* Variabel yang ada didalam kelas buku */
var $judul;
var $harga;

/*br adalah tag html untuk membuat garis baru */
function getHarga(){
echo $this->harga . "<br/>";
}

function getJudul(){
echo $this->judul . " <br/>";
}

/* Fungsi Spesial Construct didalam kelas buku */
function __construct( $parameter1, $parameter2 ){
$this->judul = $parameter1;
$this->harga = $parameter2;
}
}

Class Komik extends Buku{
var $penerbit;

Function setPenerbit($parameter){
$this->penerbit = $parameter;
}

Function getPenerbit(){
echo $this->penerbit. "<br />";
}

/* Fungsi Spesial Construct didalam kelas Komik */
function __construct( $parameter1, $parameter2, $parameter3 ){
$this->judul = $parameter1;
$this->harga = $parameter2;
$this->penerbit = $parameter3;
}
}

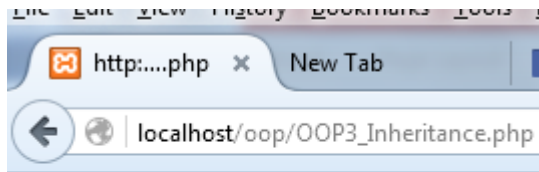
/* Buat sebuah object komikjenaka dari kelas komik */
$komikjenaka = new Komik( "Komik si juki", 85000, "juki" );

/* Memanggil Method pada kelas komik */
$komikjenaka->getJudul();
$komikjenaka->getHarga();
$komikjenaka->getPenerbit();
```



?>

Maka jika dieksekusi hasilnya adalah :



Komik si juki  
85000  
juki

Sample Full Code ada pada file zip yang telah didownload. Nama File :  
OOP3\_ConstructFunction.php

Table KRUNA 4	
Apa anda sudah membacanya <i>Chapter 6</i> secara terstruktur?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>object</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu kelas dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>Method</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>Variable</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda sudah faham bagaimana cara membuat sebuah <i>object</i> menggunakan kelas yang anda buat?	Ya / Tidak
Apa anda sudah faham bagaimana cara memanggil <i>Method</i> dalam sebuah <i>object</i> yang anda buat?	Ya / Tidak
Apa anda sudah faham bagaimana cara menggunakan fungsi <i>construct</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>Construction Function</i> ?	Ya / Tidak
Apa anda sudah faham apa itu <i>Inheritance</i> ?	Ya / Tidak
Apa anda sudah faham apa itu <i>Parent Class</i> dan <i>Child Class</i> ?	Ya / Tidak
Apa anda sudah faham bagaimana cara melakukan <i>Inheritance</i> pada <i>Class</i> yang anda buat sebelumnya?	Ya / Tidak
Tutup Halaman hafalkan kembali semua yang telah anda pelajari dari chap 1 - 6	Hafal / Tidak

## Chapter 7 : Apa itu *Function Overriding*?

Sebelumnya kita telah memahami bahwa pada *child class* yang kita buat didalamnya juga terdapat *variable* dan *method* yang berasal dari *parent class*. Lalu apa yang terjadi jika kita membuat *method* pada *child class* dengan nama yang sama seperti pada nama *method* yang ada pada *parent class*? Itulah yang disebut dengan ***Function Overriding***.

*Method* baru yang kita buat pada *child class* jika memiliki kesamaan nama dengan *method* yang ada pada *parent class* maka dia akan menimpa *method* yang berasal dari *parent class*. Agar lebih faham perhatikan *Class* komik dibawah ini :

```
Class Komik extends Buku{
    var $penerbit;

    Function setPenerbit($parameter){
        $this->penerbit = $parameter;
    }

    Function getPenerbit(){
        echo $this->penerbit. "<br />";
    }

    /* Fungsi Spesial Construct didalam kelas Komik */
    function __construct( $parameter1, $parameter2, $parameter3 ){
        $this->judul = $parameter1;
        $this->harga = $parameter2;
        $this->penerbit = $parameter3;
    }
}
```

Pada *class* Komik diatas terdapat *method* *getHarga* dan *getJudul* dikarenakan menerima turunan (*Inheritance*) dari *Class* Buku (lihat *method* pada *Class* Buku). Hanya saja tidak terlihat dikarenakan inilah kelebihan Teknik *Inheritance* dalam OOP.

```
function getHarga(){
    echo $this->harga . "<br/>";
}

function getJudul(){
    echo $this->judul . " <br/>";
}
```

Kemudian jika kita ingin membuat sebuah *method* baru didalam *Class* Komik dengan nama *method* yang sama kita bisa melakukannya, hanya saja *method* yang ada pada *Class* Buku akan tertimpa dengan *method* yang anda buat di dalam *Class* Komik. Kita langsung saja praktekan menggunakan *code* dibawah ini :

```
Class Komik extends Buku{
var $penerbit;

/*Nilai Harga dikurangi 5000 */
function getHarga(){
    echo $this->harga-5000 ."<br/>";
}

Function setPenerbit($parameter){
    $this->penerbit = $parameter;
}

Function getPenerbit(){
    echo $this->penerbit. "<br />";
}

/* Fungsi Spesial Construct didalam kelas Komik */
function __construct( $parameter1, $parameter2, $parameter3 ){
    $this->judul = $parameter1;
    $this->harga = $parameter2;
    $this->penerbit = $parameter3;
}
}
```

Pada *code* diatas kita memasukan *code* dibawah ini kedalam *Class* Komik :

```
/*Nilai Harga dikurangi 5000 */
function getHarga(){
    echo $this->harga-5000 ."<br/>";
}
```

Ini Artinya *method* dibawah ini(*code* tidak terlihat karena *inheritance* dari *Class* Buku) yang ada didalam *Class* Komik akan tertimpa oleh *code method* diatas :

```
/*br adalah tag html untuk membuat garis baru */
function getHarga(){
    echo $this->harga ."<br/>";
}
```

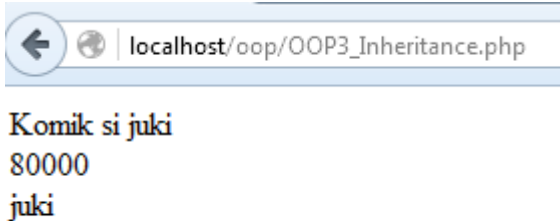
Perbedaanya hanya sederhana *method* yang baru akan mengurangi harga yang dimasukan sebesar 5000. Untuk mengujinya kita buat sebuah objek kembali :

```
/* Buat sebuah object komikjenaka dari kelas komik */
$komikjenaka = new Komik( "Komik si juki", 85000, "juki" );
```

Kemudian untuk mengeksekusinya kita perlu memanggil *method* yang ada didalam object. Masukan *code* dibawah ini :

```
/* Memanggil Method pada kelas komik */  
$komikjenaka->getJudul();  
$komikjenaka->getHarga();  
$komikjenaka->getPenerbit();
```

Jika dieksekusi hasilnya adalah :



Komik si juki  
80000  
juki

Tipe *data* integer 8500 tadi akan dikurangi 500 ini artinya *method* yang baru pada *class* *Komik* telah melakukan ***Overriding Method***.

Table KRUNA 5	
Apa anda sudah faham dan hafal segala hal yang ada didalam Table KRUNA 4 buka lagi)?	Ya / Tidak
Apa anda sudah faham apa itu <i>Function Overriding</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda sudah faham bagaimana cara melakukan <i>Function Overriding</i> pada <i>Child Class</i> yang telah anda buat?	Ya / Tidak
Apa anda sudah melakukan Studi Kasus Belajar Salah dari chap 5 sampai chap 7	Ya / Tidak
Tutup Halaman cek kembali Apa anda sudah hafal apa itu <i>Function Overriding</i> ?	Ya / Tidak

## Chapter 8 : Apa itu Access Modifiers?

Didalam bahasa pemrograman PHP jika kita ingin menggunakan konsep OOP kita akan mengenal sebuah *Access Modifiers* melalui tiga *keyword* yaitu :

1. *Public*
2. *Private*
3. *Protected*

Ketiga *keyword* tersebut digunakan didalam sebuah *Class* yang bisa kita gunakan untuk mengubah sifat akses pada suatu *variable* dan *method*. Penulis ingatkan kembali secara teknis *coding*, ***variable* disebut dengan *property* dan *method* disebut dengan *function***. Baca dan fahami *table* dibawah ini sebelum kita akan lebih memahaminya dengan praktek *Coding* :

Public	Private	Protected
Dalam PHP <i>Method</i> dan <i>variable</i> yang sudah diset menjadi <i>public</i> dapat diakses dimana saja.	Dalam PHP <i>Method</i> dan <i>variable</i> yang sudah diset menjadi <i>private</i> hanya dapat diakses oleh <i>Class</i> itu sendiri	Dalam PHP <i>Method</i> dan <i>variable</i> yang sudah diset menjadi <i>protected</i> dapat diakses oleh <i>Class</i> itu sendiri dan <i>Class</i> turunanya

Ada yang harus kita ingat jika kita ingin membuat *access modifier* pada sebuah *class* khusus untuk membuat suatu *variable* kita sudah tidak lagi menggunakan ***keyword Var***. PHP memang memiliki perbedaan dengan bahasa pemrograman *desktop* dalam menentukan tipe *data* untuk membuat sebuah *variable*. Jika ingin memahaminya lebih dalam silahkan mengunjungi :

[www.w3schools.com/php/php\\_datatypes.asp](http://www.w3schools.com/php/php_datatypes.asp)

Kecuali anda sudah faham langsung saja selanjutnya kita akan mempelajari *Access Modifier PUBLIC* menggunakan *code* dibawah ini :

```
class Buku
{
    /* Variabel yang ada didalam kelas buku */
    public $judul;

    Public function getJudul()
    {
        echo $this->judul . " <br/>";
    }
}
```

Pada *code* diatas kita bisa melihat *variable/property* *judul* saya beri *access modifier public* dan *method/function* *getJudul* saya beri *access modifier public*. Selanjutnya kita akan membuat sebuah *object* menggunakan *class* *Buku* :

```
$juki = new buku();
```

*Object* bernama *juki* ini mencoba mengakses *variable/property* judul yang ada didalam dirinya sendiri dan mengisikan sebuah nilai (string) kedalamnya :

```
$juki->judul="Tinjau Cepat OOP dengan PHP";
```

Untuk melihat isi nilai yang telah kita masukan sebelumnya, kita bisa mengakses *variable/property* judul yang ada didalam *object* *juki* secara langsung menggunakan *code* dibawah ini :

```
echo $juki->judul;
```

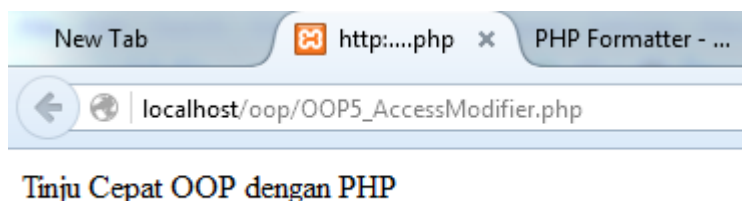
*Full Script* ;

```
<?php
class Buku
{
    /* Variabel yang ada didalam kelas buku */
    public $judul;

    public function getJudul()
    {
        echo $this->judul . " <br/>";
    }
}

/* Buat sebuah object juki dari kelas buku */
$juki = new buku();
$juki->judul="Tinjau Cepat OOP dengan PHP";
echo $juki->judul;
?>
```

Jika *script* dieksekusi maka hasilnya :



Karena *method/function* *getJudul* diberi *access modifier public* maka kita juga bisa mengakses *method/functionnya* secara langsung dengan *code* dibawah ini

```
echo $juki->getJudul();
```

Ada dua hal yang telah kita pelajari disini yaitu mengakses *variable/property* dan *method/function* yang telah diberi *access modifier public*. Segalanya bisa diakses.

Selanjutnya kita akan mempelajari *Access Modifier PRIVATE* menggunakan *code* dibawah ini :

```
<?php
class Buku
{
    /* Variabel yang ada didalam kelas buku */
    private $judul;

    public function setJudul($namajudul)
    {
        $this->judul = $namajudul;
    }

    public function getJudul()
    {
        echo $this->judul . " <br/>";
    }
}

/* Buat sebuah object juki dari kelas buku */
$juki = new buku();

/* Mengakses Method/Function setJudul*/
$juki->setJudul("Access Modifier di PHP");

/* Mengakses Method/Function getJudul*/
$juki->getJudul();
?>
```

Pada *code* diatas kita membuat *access modifier* pada *variable/property* judul menjadi *private* dan membuat sebuah fungsi *setJudul* yang telah kita beri *access modifier public* untuk menyimpan nilai kedalam *variable/property* judul.

Jika *code* diatas dieksekusi maka hasilnya :



Access Modifier di PHP

Meskipun *variable/property* judul memiliki atribut *access modifier private* kita tetap bisa mengaksesnya melalui *method/function* *getJudul*, karena *method/function* *getJudul* memiliki atribut *access modifier public*. Tapi apa yang terjadi jika kita mengakses *variable/property* judul secara langsung untuk mengetahui nilainya menggunakan *code* dibawah ini ?

```
$juki->judul;
```

Maka hasilnya adalah error seperti yang ada pada pesan dibawah ini :

**Fatal error:** Cannot access private property Buku::\$judul in C:\xampp\htdocs\oop\OOP6\_AccessModifierPrivate.php on line 21

Arti pesan diatas adalah kita tidak bisa mengakses *property(variable)* judul dalam kelas buku dikarenakan *variable/property* judul memiliki *attribute access modifier private*. Maka sudah jelas *Access Modifier Private* tidak akan bisa diakses dari luar *Class*. Disinilah fungsi atribut *private* dimanfaatkan untuk kepentingan lebih lanjut ada beberapa, *variable/property* dan *method/function* yang tidak bisa diakses untuk pengolahan *data* atau instruksi tertentu.

Kajian *Access Modifier* memang sangat sulit untuk dijelaskan tetapi setelah kita memahaminya ini akan sangat berguna kedepanya. Sedikit bermain logika menggunakan *code* dibawah ini :

```
<?php
class Buku
{
    /* Variabel yang ada didalam kelas buku */
    private $judul;
    public $test;

    public function setJudul($namajudul)
    {
        $this->judul = $namajudul;
        $this->test = $this->judul;
    }

    private function getJudul()
    {
        echo $this->test . " <br/>";
    }
}

/* Buat sebuah object juki dari kelas buku */
$juki = new buku();

/* Mengakses Method/Function setJudul*/
$juki->setJudul("Access Modifier di PHP");
echo $juki->test;

?>
```



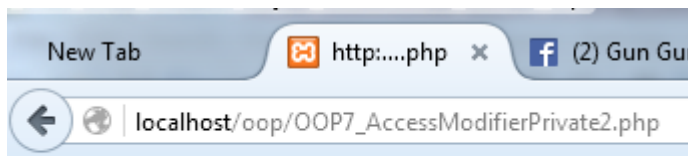
Pada *code* diatas saya membuat *variable/property* baru bernama **test** dengan *access modifier* *public*. Pada *method/function* *setJudul*, nilai yang akan masuk melalui *parameter* *\$namajudul* akan tersimpan kedalam *variable/property* *judul* yang bersifat *private* kemudian pada *statement* selanjutnya saya mencoba mencuri nilai yang ada didalam *variable/property* *judul* yang bersifat *private* untuk dimasukan kedalam *variable/property* *test* yang bersifat *public*:

```
public function setJudul($namajudul)
{
    $this->judul = $namajudul;
    $this->test = $this->judul;
}
```

Harapan saya sederhana apakah saya tetap bisa melihat nilai yang ada didalam *test* dengan mengaksesnya secara langsung karena dia bersifat *public*? Dan menggunakan *statement code* dibawah ini saya mencoba mengaksesnya :

```
echo $juki->test;
```

Setelah *code* dieksekusi saya bisa mengaksesnya :



### Access Modifier di PHP

Berbeda jika kita mencoba memaksa mengaksesnya melalui *method/function* *getJudul* yang memiliki *attribute access modifier* *private* untuk mengetahui isi nilai pada *variable/property* *test* menggunakan *statement code* dibawah ini :

```
/* Mengakses Method/Function getJudul*/
$juki->getJudul();
```

Maka jika dieksekusi hasilnya adalah :

**Fatal error:** Call to private method Buku::getJudul() from context " in C:\xampp\htdocs\oop\OOP7\_AccessModifierPrivate2.php on line 28

Maka sudah jelas sebuah *variable/property* dan *method/function* yang diberi *attribute access modifier* *private* tidak akan bisa diakses dari luar *class* dia hanya bisa diakses oleh *class* itu sendiri.

Selanjutnya kita akan mempelajari *Access Modifier* **PROTECTED** menggunakan *code* dibawah ini :

```
class Buku
{
    /* Variabel yang ada didalam kelas buku */
    public $judul;
    protected $editor;

    public function getJudul()
    {
        echo $this->judul . " <br/>";
    }

    public function getEditor()
    {
        echo $this->editor . " <br/>";
    }
}
```

Pertama kita akan membuat *class* Buku dengan *variable/property* baru yaitu **\$editor**, selanjutnya *variable/property* \$editor kita beri *attribute access modifier protected*. Ini artinya *variable/property* \$editor hanya bisa diakses oleh *class* itu sendiri dan *class* turunannya. Kemudian kita akan membuat sebuah *child class* yang menerima *inheritance* dari *Class* Buku. Menggunakan *code* dibawah ini :

```
class komik extends buku
{
    public $harga;

    function __construct($parameter1, $parameter2, $parameter3 )
    {
        $this->judul = $parameter1;
        $this->harga = $parameter2;
        $this->editor = $parameter3;
    }
}
```

Disini secara jelas kita bisa memahami *Access Modifier Protected*, sebab *variable/property* editor bisa langsung diakses oleh *class* keturunannya yaitu *class* Komik. Jika *variable/property* editor memiliki *access modifier private* maka dia tidak akan bisa diakses oleh *class* keturunannya sebab dia hanya bisa diakses oleh *class* itu sendiri (*parent class*). Jika anda lupa fungsi *construct* silahkan kembali ke chapter 5.

Penjelasan *code* diatas adalah saat kita membuat sebuah *object* menggunakan *class* *komik* ada 3 *parameter* yang bisa langsung kita masukan untuk mengisi tiga buah *variable/property*, salah satunya adalah *variable/property* editor yang telah kita beri *access modifier protected*. Selanjutnya kita akan membuat sebuah *object* baru menggunakan *code* dibawah ini :

```
/* Buat sebuah object juki dari kelas buku */  
$juki = new komik("Tinjau Cepat OOP dengan PHP",90000,"Christa Agung Winarno");
```

Apa yang terjadi jika kita mencoba mengakses *variable/property* editor secara langsung menggunakan *code* dibawah ini ?

```
/* Mengakses variable/property */  
echo $juki->editor . " <br/>";
```

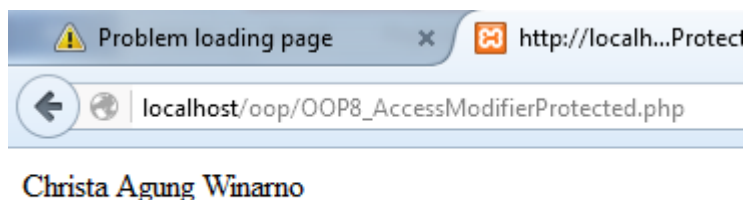
Hasilnya adalah Error :

**Fatal error:** Cannot access protected property *komik::\$editor* in *C:\xampp\htdocs\oop\OOP8\_AccessModifierProtected.php* on line 31

Arti pesan *error* diatas adalah kita tidak bisa mengakses *variable/property editor* pada *class* *komik* secara langsung karena ia memiliki *access modifier protected*. Kita bisa mengakses isi nilai *variable/property* editor melalui *method/function* *getEditor* karena dia telah diberi *access modifier public* :

```
/* Mengakses Method/Function */  
echo $juki->getEditor();
```

Jika dieksekusi hasilnya adalah :



Sample Full *Code* ada pada *file* zip yang telah didownload. Nama *File* : *OOP8\_AccessModifierProtected.php*

Table KRUNA 6	
Apa anda sudah faham dan hafal segala hal yang ada didalam Table KRUNA 4 & 5 buka lagi)?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>Function Overriding</i> dalam sebuah <i>object Oriented Programming</i> ?	Ya / Tidak
Apa anda masih ingat dan faham bagaimana cara melakukan <i>Function Overriding</i> pada <i>Child Class</i> yang telah anda buat?	Ya / Tidak
Apa anda sudah faham apa itu <i>Access Modifier</i> pada <i>Object Oriented Programming</i> ?	Ya / Tidak
Apa anda sudah faham perbedaan atribut <i>public</i> , <i>private</i> dan <i>protected</i> ?	Ya / Tidak
Apa anda sudah melakukan Studi Kasus Belajar Salah dari chap 8?	Ya / Tidak
Tutup Halaman cek kembali Apa anda sudah hafal apa itu <i>Access Modifier</i> ?	Ya / Tidak

## Read Control

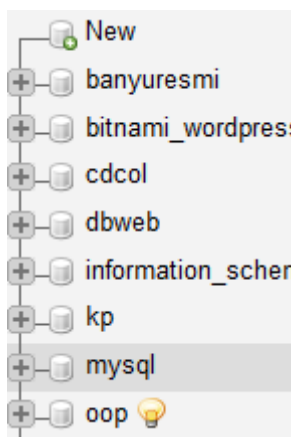
Ini adalah sebuah halaman khusus yang menjadi tanda bahwa kita telah selesai mempelajari kajian OOP dengan PHP, seluruh kajian yang telah kita bahas dari chapter 1 sampai *chapter 8* akan kita gunakan di *chapter* selanjut. Anda harus memverifikasi terlebih dahulu menggunakan table KRUNA bahwa anda telah hafal dan faham segala kajian yang ada di *chapter 1* sampai *chapter 8*. Sebab jika belum anda akan kesulitan memahami bab selanjutnya.

Pada *chapter* selanjutnya kita sudah mulai untuk berinteraksi dengan *database* menggunakan PHP, tentunya PHP dengan *style* OOP. Menggunakan *Class* PDO yang kita gunakan untuk berinteraksi dengan *database*, Menggunakan *method/function* yang ada pada *Class* PDO untuk mengeksekusi perintah-perintah *SQL* dan memahami *result set*.

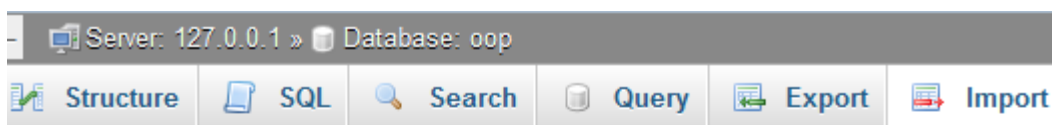
## Chapter 9 : Apa itu PDO (PHP Data Object) ?

PDO adalah bagian dari asset PHP yang digunakan untuk berinteraksi dengan *database*. Untuk berinteraksi dengan *database* PDO mendukung banyak sekali *database* populer yang bisa digunakan melalui PDO. Berbeda dengan *method/function mysqli\_connect()*, *method* ini hanya bisa digunakan untuk melakukan koneksi ke *database* mysql saja. Sementara PDO mendukung lebih dari 5 *database* yaitu **MySQL, SQLite, Firebird, Oracle, ODBC, IBM** dan masih banyak lagi.

Sebelum kita melanjutkan kajian PDO ada beberapa hal yang harus kita persiapkan, pertama silahkan anda masuk ke halaman phpmyadmin untuk membuat sebuah *database*. Beri nama *database* tersebut oop seperti pada gambar dibawah ini :



Setelah kita membuat *database* OOP selanjutnya kita *import script* SQL yang telah saya sediakan didalam zip yang telah anda *download*, pilih tab *import* kemudian klik *browse* :



### Importing into the database "oop"

#### File to Import:

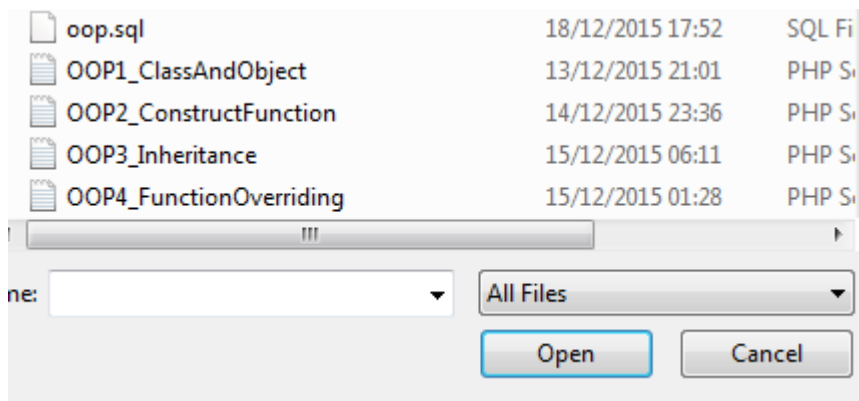
File may be compressed (gzip, bzip2, zip) or uncompressed.

A compressed file's name must end in **[format].[compression]**. Example: **.sql.zip**

Browse your computer:  No file selected. (Max: 2,048KiB)

Character set of the file:

Selanjutnya cari *file* bernama oop.sql kemudian pilih open :



Jika *import* telah selesai tekan tombol go dibagian paling bawah:

### Format-Specific Options:

SQL compatibility

☒ Do not use ANSI

Go

Jika berhasil maka akan muncul sebuah *table* bernama *table1* yang berisi 3 *record* :

id	nama	email
1	Gun Gun Febrianza	gungunfebrianza@email.com
2	Christa Agung Winarno	CAW@email.com
3	Maudy Ayunda	mod@email.com

Selanjutnya buatlah sebuah *folder* bernama *oop* di :

C:\xampp\htdocs\oop

Didalam *folder* *oop* buat lagi sebuah *folder* beri nama *lib* dan didalamnya buat sebuah *file* *php* bernama *crudclass.php*, sekarang persiapan sudah selesai lets coding!

Sekarang didalam *mainlib.php* kita akan membuat *object* menggunakan *Class* milik PDO yang sudah tersedia didalam PHP (PDO sudah tersedia pada versi PHP 5.1 keatas) menggunakan *statement code* dibawah ini :

```

<?php
try
{
$con = new PDO("mysql:host=localhost;dbname=oop","root","");
}
catch(PDOException $e)
{
    echo $e->getMessage();
}
?>

```

Perhatikan *parameter* pada constructor dalam *class* PDO dibawah ini :

New PDO (\$DSN, \$USERNAME, \$PASSWORD)

Pada *statement code* diatas kita mencoba membuat sebuah *object* menggunakan *class* PDO berikut dengan 3 *Parameter* yang harus kita isi. DSN adalah singkatan dari **Data Source Name** ini mengacu kepada *database* apa yang kita gunakan dan nama *database* yang akan kita koneksikan, **Username** dan **Password** adalah **user credential** yang kita butuhkan untuk mengakses sebuah *database*.

Pada *code* diatas *Object* \$con dibuat, pada *parameter* pertama terdapat *Data Source Name* menggunakan *database* MySQL, dalam *localhost* dan nama *database* oop. Pada *parameter* kedua root adalah nama *user* dan *password* kosong.

Setelah kita membuat sebuah PDO *object* selanjutnya kita bisa memanggil seluruh kemampuan yang dimilikinya. Kemampuan itu adalah *method/function* yang dimiliki oleh *Class* PDO. Hal ini akan kita bahas di *chapter* selanjutnya. ☺

Table KRUNA 7	
Apa anda masih faham dan hafal segala hal yang ada didalam Table KRUNA 4, 5 dan 6 (buka lagi)?	Ya / Tidak
Apa anda tau apa itu PDO?	Ya / Tidak
Bagaimana cara membuat sebuah <i>object</i> menggunakan <i>Class</i> PDO?	Ya / Tidak
Apa itu <i>Data Source Name</i> dan <i>User Credential</i> ?	Ya / Tidak
Apa anda sudah melakukan Studi Kasus Belajar Salah dari chap 9?	Ya / Tidak



## Chapter 10 : *Method Query()*

*Method query()* adalah salah satu *method/function* yang dimiliki *class* PDO untuk mengeksekusi sebuah **Select Statement**. Jika *method/function* ini digunakan maka akan mengembalikan (*return*) sebuah *PDOstatement object* yang didalamnya terdapat sebuah hasil (*result set*) yang terdiri dari baris dan kolom, jika tidak terdapat sebuah hasil maka dia akan mengembalikan nilai logika **FALSE**.

Agar lebih faham buatlah *file* bernama *pdo\_select.php* didalam *folder* *oop*. Kemudian masukan *code* dibawah ini :

```
<?php
include('lib/crudclass.php');

$result = $con->query('SELECT * FROM table1');

foreach ($result as $d) :
echo $d['nama'] . " <br/>";
endforeach;
?>
```

Pada *code* diatas kita mencoba mengeksekusi sebuah *sql statement* sederhana untuk melihat seluruh isi kolom (*column*) dan baris (*row*) yang ada didalam *table1* menggunakan *method/function query*. Return dari *method query* adalah seluruh *column* dan *row* yang ada didalam *table1* yang kemudian disimpan didalam *variable* **\$result**, selanjutnya kita menyebut *variable* **\$result** sebagai **PDOStatement Object**.

Kemudian untuk melihat isi **PDOstatement Object \$result** kita bisa menggunakan *control structure foreach* dan hanya mengambil *column* yang memiliki string '**nama**' dan seluruh isi *record* yang ada didalamnya. Jika dieksekusi maka hasilnya :



Gun Gun Febrianza  
Christa Agung Winarno  
Maudy Ayunda

## Chapter 11 : Method Exec()

Untuk mengeksekusi perintah INSERT, UPDATE dan DELETE menggunakan **PDO Object** kita harus menggunakan *method/function* exec. Ketika *method* ini dieksekusi maka akan mengembalikan (*return*) jumlah baris (*rows*) yang telah dieksekusi oleh suatu *statement* SQL jika tidak ada jumlah baris yang dieksekusi maka akan mengembalikan nilai 0. Agar lebih faham kita akan mengeksekusi perintah **INSERT** buatlah *file* bernama pdo\_insert.php didalam *folder* oop. Kemudian masukan *code* dibawah ini :

```
<?php
include('lib/crudclass.php');
$result = $con->exec("INSERT INTO table1 (nama, email) VALUES ('mod',
'mod@gmail.com')");
echo $result;
?>
```

Pada *code* diatas *Object* **\$con** memanggil *method/function* exec berikut dengan SQL *statement* untuk melakukan **INSERT data**. Jika *code* diatas berhasil dieksekusi maka akan menghasilkan sebuah *return* jumlah baris yang berhasil dieksekusi seperti pada gambar dibawah ini :



1

Jika kita melihat kembali table1 yang telah kita buat maka akan muncul *record* baru :

		id	nama	email
<input type="checkbox"/>	Edit  Copy  Delete	1	Gun Gun Febrianza	gungunfebrianza@email.com
<input type="checkbox"/>	Edit  Copy  Delete	2	Christa Agung Winarno	CAW@email.com
<input type="checkbox"/>	Edit  Copy  Delete	3	Maudy Ayunda	mod@email.com
<input type="checkbox"/>	Edit  Copy  Delete	4	mod	mod@gmail.com

Sekarang kita akan mencoba mengeksekusi perintah UPDATE, buatlah *file* bernama pdo\_update.php didalam *folder* oop. Kemudian masukan *code* dibawah ini :

```
<?php
include('lib/crudclass.php');
$result = $con->exec("UPDATE table1 SET nama = 'Ayunda Faza Maudya' WHERE id =
3 ");
echo $result;
?>
```

Pada *code* diatas *Object* **\$con** memanggil *method/function* *exec* berikut dengan *SQL statement* untuk melakukan **UPDATE** *data*. Arti perintah *SQL* diatas adalah kita akan mengubah *record* pada *column* **nama** yang memiliki id 3. Jika *code* diatas berhasil dieksekusi maka akan menghasilkan sebuah *return* jumlah baris yang berhasil dieksekusi seperti pada gambar dibawah ini :



1

Jika kita melihat kembali *table1* akan ada *record* yang berubah pada **id** nomor 3 :

+ Options

				id	nama	email
<input type="checkbox"/>	Edit	Copy	Delete	1	Gun Gun Febrianza	gungunfebrianza@email.com
<input type="checkbox"/>	Edit	Copy	Delete	2	Christa Agung Winarno	CAW@email.com
<input type="checkbox"/>	Edit	Copy	Delete	3	Ayunda Faza Maudya	mod@email.com
<input type="checkbox"/>	Edit	Copy	Delete	4	mod	mod@gmail.com

Sekarang kita akan mencoba menghapus *record* pada *table1* yang memiliki nilai *id* = 4, sebelum kita akan mengeksekusi perintah **DELETE** buatlah *file* bernama *pdo\_delete.php* didalam *folder* *oop* kemudian masukan *code* dibawah ini :

```
<?php
include('lib/crudclass.php');
$result = $con->exec("DELETE FROM table1 WHERE id = 4 ");
echo $result;
?>
```

Jika *code* diatas dieksekusi maka *record* dengan *id* 4 akan terhapus.

Table KRUNA 8	
Apa anda masih faham dan hafal segala hal yang ada didalam Table KRUNA 4, 5, 6 dan 7 (buka lagi)?	Ya / Tidak
Apa anda masih ingat dan faham apa itu PDO?	Ya / Tidak
Apa anda masih ingat dan faham bagaimana cara membuat sebuah <i>object</i> menggunakan <i>Class</i> PDO?	Ya / Tidak
Apa anda masih ingat dan faham Apa itu <i>Data Source Name</i> dan <i>User Credential</i> ?	Ya / Tidak
Apa anda tahu apa itu <i>method Query</i> ?	Ya / Tidak
Apa anda tahu apa itu <i>method exec</i> ?	Ya / Tidak
Apa anda tahu apa itu <i>PDO Statement Object</i> ?	Ya / Tidak
Apa anda tahu apa itu <i>return</i> ?	Ya / Tidak
Tutup halaman ini apa anda sudah yakin memahami <i>chapter</i> 11?	Ya / Tidak
Apa anda sudah melakukan Studi Kasus Belajar Salah dari chap 10 - 11?	Ya / Tidak

## Chapter 12 : Try/Catch Statement

Sewaktu waktu kita akan berhadapan dengan beberapa kasus yang mungkin kita gagal membuat sebuah *object* dari *class* PDO atau mengakses *method/function* yang dimiliki oleh *object* PDO. Jika hal itu terjadi maka *Class* PDO akan mengeluarkan **Throws** (mekanisme khusus ketika terjadi sebuah kegagalan) yang dikenal dengan sebutan **exception**.

**Exception** itu sendiri dia adalah sebuah *object* yang mengandung informasi tentang segala *error* yang terjadi. Jika *exception* tidak diciptakan maka program yang kita buat akan berhenti alias gagal untuk mengeksekusi baris *code* selanjutnya. Untuk mengatasi hal ini kita bisa menggunakan *try/catch statement* yang mungkin disadari atau tanpa anda sadari di *chapter* (chapter 9) sebelumnya saya memasukannya tanpa menjelaskannya sedikitpun. Agar pembaca bisa fokus satu persatu.

Contoh kasus yang terjadi kita bisa mempelajari *code* dibawah ini :

```
try {
    $db = new PDO($dsn, $username, $password);
    echo 1<p>You are connected to the database !</p>;
} catch (PDOException $e) {
    $error_message = $e->getMessage(); echo "<p>An error occurred while connecting to the
    database: $error_message </p>";
}
```

Pada *code* diatas misalkan kita membuat sebuah *object* *\$db* menggunakan *class* PDO namun kita salah memasukan *Data Source Name*, user dan *password* (atau salah satunya) maka *object* akan gagal dibuat tetapi karena dia berada didalam *curly bracket* (*{}*) *keyword* *try* maka program tidak akan langsung berhenti melainkan informasi error akan disimpan pada *object* *\$e* selanjutnya kita membuat *variable* bernama **\$error\_message** yang akan digunakan untuk menerima informasi **error** dari *object* *\$e*. *Object* *\$e* atau **exception** ini memanggil *method/function* **getMessage()** yang ada didalam *class* PDO agar ia bisa mendapatkan pesan *error* dan menyimpan informasinya ke *variable* *\$error\_message*.

Jadi sangat disarankan kedepanya ketika kita akan mengeksekusi sebuah *statement code*, membuat sebuah *object*, memanggil *method/function* kita menyimpan *statement code* tersebut didalam *curly bracket* **keyword** *try* :

```
Try{

Masukan statements code anda disini

} catch (PDOException $e) {

$error_message = $e->getMessage();
echo $error_message;

}
```

## Chapter 13 : Memahami Array

Dari chapter 10 sampai chapter 11 kita sudah mencoba melakukan perintah SQL *statements* dasar seperti SELECT, INSERT, UPDATE dan Delete dengan memanfaatkan *method/function query* dan *exec* pada *object* yang kita buat dari *Class PDO*. Tetapi sebenarnya belum sempurna jika kita ingin berinteraksi dengan *database* tanpa memahami sebuah *Array* dan cara Kerjanya. Selanjutnya kita akan lebih memahami bagaimana cara mengambil sebuah *data* dalam result set yang tersimpan dalam sebuah *PDOstatement Object*.

Sebuah *array* bisa menyimpan satu atau lebih *element* didalamnya. *Array* adalah sebuah *variable special* yang bisa menampung banyak *data* dalam waktu yang bersamaan. Untuk membuat sebuah *array* kita bisa menggunakan *function* :

```
Array();
```

Didalam PHP ada 3 tipe *Array* namun yang akan kita bahas hanya dua saja yaitu

1. Indexed Array
2. Associative Array
3. Multidimensional Array (tidak akan dibahas)

Selanjutnya kita akan mempelajari bagaimana cara melakukan set/get sebuah *array* menggunakan *Indexed Array*. buatlah *file* bernama OOP9\_IndexedArray.php didalam *folder* oop kemudian masukan *code* dibawah ini :

```
<?php
/*set (membuat) sebuah array*/
$motor = array("Ninja","CBR","R15","Tiger")
?>
```

Maksud *statement code* diatas adalah **\$motor** adalah sebuah *Indexed Array* yang didalamnya terdapat 3 elemen, tiga karena kita mulai menghitungnya dari 0. Jika direpresentasikan kedalam sebuah *table* kira kira bentuknya seperti ini :

Index	Element
0	Ninja
1	CBR
2	R15
3	Tiger

Kemudian jika kita ingin mendapatkan *data* yang kita inginkan dari sebuah *array* kita bisa menggunakan index untuk memanggilnya, misalkan kita ingin mendapatkan *string R15* :

```
<?php
/*set (membuat) sebuah array*/
$motor = array("Ninja","CBR","R15","Tiger");

/*get (mendapatkan) sebuah elemen dalam sebuah array*/
$elemen = $motor[2];
echo $elemen;
?>
```

Pada *statement code* **\$elemen = \$motor[2];** diatas, artinya kita mencoba memanggil nilai sebuah elemen pada array motor yang berada pada posisi index 2 dan menyimpannya kedalam *variable* **\$elemen**. Jika *code* diatas dieksekusi maka hasilnya :



Jadi sudah jelas jika kita ingin memanggil nilai elemen lainya pada *array* motor kuncinya atau keynya cukup menggunakan indexnya. Kemudian jika kita ingin menampilkan seluruh elemen yang ada didalam *array* motor ada dua cara yang bisa kita lakukan :

```
<?php
/*set (membuat) sebuah array*/
$motor = array("Ninja","CBR","R15","Tiger");

/*get (mendapatkan) sebuah elemen dalam sebuah array*/
$elemen = $motor[2];
echo $elemen . "<br>";

/*Menggunakan foreach*/
foreach ($motor as $moto)
{
    echo $moto . "<br>";
}
?>
```

Yang pertama menggunakan **keyword foreach**, *looping* akan terjadi setiap elemen yang ada didalam *array* **\$motor** akan disimpan dan dirilis satu persatu kedalam *variable* **\$moto** kemudian elemen ditampilkan seterusnya begitu sampai akhir sampai elemen yang terakhir. Jika *code* diatas dieksekusi maka hasilnya :

R15  
Ninja  
CBR  
R15  
Tiger

Kemudian yang kedua menggunakan **keyword for** sambil menggunakan **function count()** yang dimiliki oleh **array**, perhatikan **code** dibawah ini :

```
<?php
/*set (membuat) sebuah array*/
$motor = array("Ninja","CBR","R15","Tiger");

/*get (mendapatkan) sebuah elemen dalam sebuah array*/
$elemen = $motor[2];
echo $elemen . "<br>";

/*Menggunakan foreach*/
foreach ($motor as $moto)
{
    echo $moto . "<br>";
}

/*Menghitung jumlah elemen yang ada didalam sebuah array*/
$hitung = count($motor);
echo $hitung . "<br>";

/*Menggunakan for*/
for ($i=0; $i < count($motor); $i++)
{
    echo $motor[$i] . "<br>";
}
?>
```

Pada *statement code* diatas kita membuat **variable \$hitung** untuk menyimpan nilai *return* tipe *data integer* dari **method count**. **Method Count** membaca *array* \$motor dan menghitung jumlah elemennya, total semuanya ada 4 karena **method count** melakukan penghitungan dari angka 1.

Selanjutnya adalah *statement code* yang menggunakan **keyword for** untuk menampilkan seluruh elemen didalam *array* \$motor. Jika dieksekusi maka hasilnya adalah :



R15  
Ninja  
CBR  
R15  
Tiger  
4  
Ninja  
CBR  
R15  
Tiger

Selanjutnya kita akan mempelajari bagaimana cara melakukan set/get sebuah array menggunakan *Associative Array*. Perbedaananya pada *array* tipe ini keynya tidak lagi menggunakan index tapi sebuah nama.

Agar lebih faham kita buatlah *file* bernama OOP10\_AssociativeArray.php didalam *folder* oop kemudian masukan *code* dibawah ini :

```
<?php
/*set (membuat) sebuah Associative array*/
$umur = array("GGF"=>"23", "Christa"=>"37", "Maudy"=>"21");

/*get (mendapatkan) sebuah elemen dalam sebuah array*/
$elemen = $umur['GGF'];
echo $elemen . "<br>";
?>
```

Pada *statement code* diatas kita membuat sebuah **associative array** bernama **\$umur** kita bisa melihat perbedaananya keynya menggunakan nama berbeda dengan *indexed* yang cenderung menggunakan *angka*. Jika *code* diatas dieksekusi hasilnya adalah :

23

Untuk mendapatkan seluruh nilai elemen yang ada didalam *associative array* kita bisa menggunakan *keyword foreach* :

```

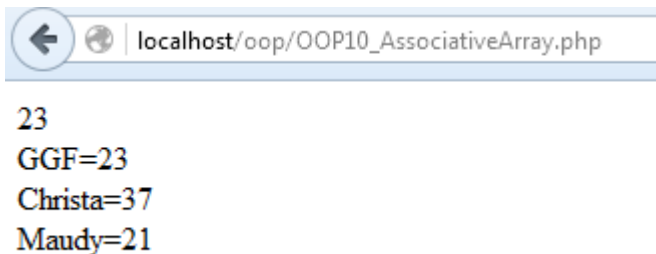
<?php
/*set (membuat) sebuah Associative array*/
$umur = array("GGF"=>"23", "Christa"=>"37", "Maudy"=>"21");

/*get (mendapatkan) sebuah elemen dalam sebuah array*/
$elemen = $umur['GGF'];
echo $elemen . "<br>";

/*Menggunakan foreach*/
foreach ($umur as $key=>$value)
{
    echo $key . "=" . $value . "<br>";
}
?>

```

Jika *code* diatas dieksekusi maka hasilnya adalah :



23  
GGF=23  
Christa=37  
Maudy=21

Table KRUNA 9	
Apa anda masih faham dan hafal segala hal yang ada didalam Table KRUNA 4, 5, 6, 7 dan 8 (buka lagi)?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>method Query</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>method exec</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>PDO Statement Object</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>return</i> ?	Ya / Tidak
Apa anda tahu apa itu <i>Array</i> ?	Ya / Tidak
Apa anda tahu apa itu <i>indexed array</i> ?	Ya / Tidak
Apa anda tahu apa itu <i>associative array</i> ?	Ya / Tidak
Tutup halaman ini apa anda sudah yakin memahami <i>Array</i> ?	Ya / Tidak
Apa anda sudah melakukan Studi Kasus Belajar Salah dari chap 12 - 13?	Ya / Tidak

## Chapter 14 : Method Fetch()

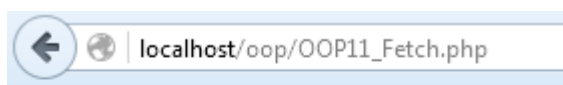
Jika kita melakukan operasi SQL menggunakan **SELECT statement** kemudian *return* yang didapatkan adalah sebuah *result set* yang kontennya hanya satu baris maka kita bisa mendapatkan *data* tersebut menggunakan **method/function fetch()** yang ada pada sebuah **PDOstatement Object**.

Agar lebih faham kita buatlah *file* bernama OOP11\_Fetch.php didalam *folder* oop kemudian masukan *code* dibawah ini:

```
<?php
include('lib/crudclass.php');
$result = $con->query('SELECT nama, email FROM table1 where id = 3');
$res = $result->fetch();
$nama = $res['nama'];
$email = $res['email'];
echo $nama . " - " . $email;
?>
```

Pada *code* diatas kita mengeksekusi sebuah **SQL statement SELECT** menggunakan *method/function query* dan mendapatkan nilai *return* sebuah **PDOStatement Object** yang didalamnya terdapat sebuah *result set*. Konten *result set* tersebut hanya satu baris kemudian hasil dari *result set* tersebut disimpan kedalam *variable* **\$result** yang selanjutnya kita menyebut *variable* ini sebagai sebuah **PDOstatement Object**.

Karena **\$result** adalah sebuah **PDOStatement Object**, dia bisa memanggil **method fetch()** dan menghasilkan *return* sebuah *array* satu baris yang disimpan kedalam *variable* **\$res**. Ini adalah alasan mengapa sebelumnya kita mempelajari *array*. Jika *code* diatas dieksekusi maka hasilnya :



Ayunda Faza Maudya - mod@email.com

Saat kita menggunakan *method fetch()* untuk mendapatkan *array* satu baris, maka *array* tersebut memiliki *string index* untuk mengakses sebuah *column* dan *numeric index* yang juga bisa digunakan untuk mengakses *column* berdasarkan posisi. Pada *code* diatas kita mengakses sebuah *column* pada *array* **\$res** menggunakan *string index*.

```
$nama = $res['nama'];
$email = $res['email'];
```

Kita juga bisa mengaksesnya menggunakan *numeric index* dengan *code* dibawah ini :

```
$nama = $res[0];  
$email = $res[1];
```

Silahkan anda mencobanya sendiri dan jangan lupa jika kita menggunakan *numeric index* kolom (*column*) pertama indexnya dimulai dari 0.

Table KRUNA 10	
Apa anda masih faham dan hafal segala hal yang ada didalam Table KRUNA 4, 5, 6,7 dan 8 (buka lagi)?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>Array</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>indexed array</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>associative array</i> ?	Ya / Tidak
Apa anda tahu apa itu <i>method fetch</i> ?	Ya / Tidak
Apa anda tahu apa itu <i>numeric index</i> dan <i>string index</i> pada sebuah <i>array</i> ?	Ya / Tidak
Tutup halaman ini apa anda sudah yakin memahami <i>chapter 14</i> ?	Ya / Tidak
Apa anda sudah melakukan Studi Kasus Belajar Salah dari chap 12 - 13?	Ya / Tidak

## Read Control

Ini adalah sebuah halaman khusus yang menjadi tanda bahwa kita telah selesai mempelajari kajian dasar PDO di PHP, seluruh kajian yang telah kita bahas dari *chapter* 9 sampai *chapter* 14 akan kita gunakan di *chapter* selanjut. Anda harus memverifikasi terlebih dahulu menggunakan table KRUNA bahwa anda telah hafal dan faham segala kajian yang ada di *chapter* 9 sampai *chapter* 14. Sebab jika belum anda akan kesulitan memahami bab selanjutnya.

Pada *chapter* selanjutnya kita akan mengenal dan memahami *prepared statement*, manfaat *prepared statement*, *method prepare*, *method execute*, *method bindParam* dan *finishing touch* segala hal yang telah kita pelajari dari *chapter* 1 dengan membuat sebuah *web application* sambil sedikit mempelajari *framework bootstrap* untuk desain ***graphic user interface web application*** yang kita buat.

## Chapter 15 : Prepared Statement

*Prepared statement* adalah sebuah fitur yang digunakan untuk mengeksekusi perintah SQL yang berulang dengan tingkat efisiensi yang tinggi. *Prepared statement* terbagi menjadi dua fase yaitu fase *prepare* dan fase *execute*.

Pada fase *prepare*, *SQL statement template* dibuat dan dikirimkan ke *Database Server*. Contoh `INSERT INTO table1 VALUES (?, ?, ?)`. Label `'?'` disebutnya **Question Mark Placeholder** adalah sebuah *parameter* yang memiliki nilai. Kemudian *database server* menganalisa *SQL statement* yang kita buat untuk di optimasi dan menyimpan hasilnya tanpa mengeksekusinya.

Pada fase *execute*, fase ini terjadi ketika program yang kita buat menyimpan nilai kedalam *parameter* `'?'` dan melakukan perintah eksekusi pada *database server*. Selanjutnya program bisa mengeksekusi *statement* berulang-ulang sebanyak mungkin dengan hanya tinggal mengubah nilai yang ada didalam *parameter* `'?'`.

Manfaatnya adalah performansi dan keamanan, secara performansi waktu *parsing sql statement* yang terjadi di *server* berkurang, meminimalisir *bandwidth* suatu server karena kita hanya mengirim nilai yang ada didalam *parameter* tanpa harus mengirimkan lagi *SQL Statement* dan secara keamanan untuk mencegah serangan SQL Injection pada *database server*.

Secara teoritisnya begitu dan secara teknisnya perhatikan *code* dibawah ini :

```
<?php
$stmt = $con->prepare("INSERT INTO table1 (nama, email) VALUES (?, ?)");
$stmt->bindParam(1, $nama);
$stmt->bindParam(2, $email);

// memasukan satu baris record dan mengeksekusinya
$nama = 'Kaiz';
$email = 'Kaiz@gmail.com';
$stmt->execute();

// memasukan lagi satu baris dan mengeksekusinya lagi
$nama = 'mamba';
$email = 'mamba@gmail.com';
$stmt->execute();
?>
```

Pada *code* diatas kita melakukan operasi *SQL statement INSERT* menggunakan *prepared statement*, selanjutnya kita bisa mengeksekusi perintah *SQL statement INSERT* tanpa harus

lagi menulis dan mengirimkan **SQL statement INSERT** berulang ke *database server* kita hanya mengirimkan nilai yang ada didalam *parameter* sehingga penggunaan *bandwidth* bisa menjadi lebih hemat.

Untuk menggunakan *prepared statement* pertama kita akan membuat koneksi ke *mysql* terlebih dahulu menggunakan *Class PDO*. Agar lebih faham buatlah sebuah *file* bernama `OOP12_PreparedStatement.php` didalam *folder oop* kemudian masukan *code* dibawah ini :

```
<?php
try
{
    $con = new PDO('mysql:host=localhost;dbname=oop', 'root', '');
} catch (PDOException $e){
    echo "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Pada *code* diatas kita membuat *object* bernama **\$con** menggunakan *class PDO* untuk dan disimpan didalam *curly bracket keyword try* untuk melakukan *error handling* jika terdapat sebuah kesalahan.

Jika kita ingin memanfaatkan ***persistent connection*** dibawah *object \$con* tambahkan *statement code* dibawah ini kedalam *parameter* berikutnya :

```
array(PDO::ATTR_PERSISTENT => true)
```

*Web* yang kita buat bisa menjadi lebih cepat karena menggunakan *persistent connection* pada *database server*. **Persistent Connection** berarti koneksi ke *database server* tidak akan ditutup meskipun sudah sampai akhir *script*, tetapi menyimpannya dalam sebuah *cache* dan menggunakannya kembali jika ada *script* lain yang ingin terhubung ke *database server* tetapi masih menggunakan *user credential* yang sama.

## Chapter 16 : Method BindParam

Dalam *prepared statement* kita bisa menyimpan sebuah nilai dalam *parameter* kedalam *variable* tertentu. Ada dua *variable* yaitu ***named placeholder*** dan ***question mark placeholder*** yang digunakan didalam *SQL statement* untuk melakukan *prepare statement*.

Sekarang kita akan mencoba mengeksekusi *prepared statement* yang menggunakan *named placeholder* kita akan menambahkan beberapa *statement code* kedalam file OOP12\_PreparedStatement.php menjadi :

```
<?php
try
{
    $con = new PDO('mysql:host=localhost;dbname=oop', 'root', '',
array(PDO::ATTR_PERSISTENT => true));

} catch (PDOException $e){
    echo "Error!: " . $e->getMessage() . "<br/>";
    die();
}

/* Mengeksekusi prepared statement dengan melakukan operasi binding variable */
$id = 3;
$sth = $con->prepare('SELECT nama, email
    FROM table1
    WHERE id = :id');
$sth->bindParam(':id', $id, PDO::PARAM_INT);
$sth->execute();
$st = $sth->fetch();
echo "Nama : " . $st['nama'] . "<br>";
echo "Email : " . $st['email'];
?>
```

Pada *code* diatas kita ***object*** **\$con** mengakses ***method*** ***prepare*** dan menyimpan ***SQL Statement*** ***SELECT*** yang akan menghasilkan *return* sebuah *record* satu baris. ***SQL Statement*** ***SELECT*** tersebut akan menjadi ***SQL Statement Template*** yang akan dikirimkan ke *database server* untuk dioptimasi dan disimpan. Selanjutnya kita hanya tinggal mengirimkan nilai nilai yang ada didalam *parameter* ke *database server*.

Kemudian pada *statement code* selanjutnya kita melakukan proses ***variable binding*** dimana nilai pada *variable* **\$id** dimasukan kedalam ***method*** ***bindParam*** untuk disimpan kedalam ***named placeholder*** :id. Selanjutnya nilai ***named placeholder*** :id pada ***method*** ***bindParam*** akan dikirimkan ke ***named placeholder*** :id yang ada di ***SQL Statement Template***.

```
$sth->bindParam(':id', $id, PDO::PARAM_INT);
```



Pada *parameter* ke 3 dari *method bindParam* adalah **PDO::PARAM\_INT** karena pada kasus ini nilai dari *variable \$id* tipe datanya adalah *integer*. Kemudian pada *statement* selanjutnya kita memanggil *method execute* untuk mengeksekusi *prepared statement* yang telah kita siapkan. Jika kode diatas dieksekusi maka hasilnya adalah :



Nama : Ayunda Faza Maudya

Email : mod@email.com

Kemudian jika kita ingin mengeksekusi lagi menggunakan *SQL statement template* diatas kita tinggal membuat *variable* yang akan di *binding* kedalam *parameter* dan menambahkan *statement code* dibawah ini :

```
<?php
try
{
    $con = new PDO('mysql:host=localhost;dbname=oop', 'root', '',
array(PDO::ATTR_PERSISTENT => true));

} catch (PDOException $e){
    echo "Error!: " . $e->getMessage() . "<br/>";
    die();
}

/* Mengeksekusi prepared statement dengan melakukan operasi binding variable */
$id = 3;
$stmt = $con->prepare('SELECT nama, email
    FROM table1
    WHERE id = :id');
$stmt->bindParam(':id', $id, PDO::PARAM_INT);
$stmt->execute();
$result = $stmt->fetch();
echo "Nama : " . $result['nama'] . "<br>";
echo "Email : " . $result['email'] . "<br>";

$id= 1;
$stmt->execute();
$result = $stmt->fetch();
echo "Nama : " . $result['nama'] . "<br>";
echo "Email : " . $result['email'];

?>
```

Jika *code* diatas dieksekusi maka hasilnya adalah :

Nama : Ayunda Faza Maudya

Email : mod@email.com

Nama : Gun Gun Febrianza

Email : gungunfebrianza@email.com

Sekarang kita akan mencoba mengeksekusi *prepared statement* yang menggunakan *Question Mark Placeholder*. Tambahkan *statement code* dibawah ini kedalam file OOP12\_PreparedStatement.php

```
/*Prepared statement menggunakan variable Question Mark Placeholder*/
$id = 2;
$email = 'CAW@email.com';
$stmt = $con->prepare('SELECT nama
    FROM table1
    WHERE id = ? AND email = ?');
$stmt->bindParam(1, $id, PDO::PARAM_INT);
$stmt->bindParam(2, $email, PDO::PARAM_STR, 15);
$stmt->execute();
$result = $stmt->fetch();
echo "<br><br> Nama : " . $result['nama'] . "<br>";
```

Pada *code* diatas kita **object** *\$con* mengakses **method** *prepare* untuk mengeksekusi *SQL Statement SELECT* dengan syarat id dan email harus sama. Didalam *SQL Statement* diatas terdapat sebuah *parameter* “?” yang disebut dengan *Question Mark Placeholder*, sebuah tempat yang akan digunakan untuk menerima nilai.

Kemudian pada *statement code* selanjutnya kita melakukan proses **variable binding** hanya saja untuk QMP (*Question Mark Placeholder*), *parameter* pertama dari **method** *bindParam* adalah angka 1 ini artinya mengacu pada “?” pertama yang ada didalam *SQL Statement Template* yaitu untuk id dan pada *parameter* kedua kita melihat *variable* *\$id* sebagai *variable* yang hendak di *binding* ke angka 1 dengan tipe data *integer*.

Kemudian pada *statement code* selanjutnya **method** *bindParam* dipanggil lagi pada *parameter* pertama nilainya angka 2 ini artinya mengacu pada “?” pertama yang ada didalam *SQL Statement Template* yaitu untuk *email* dan pada *parameter* kedua kita melihat *variable* *\$email* sebagai *variable* yang hendak di *binding* ke angka 2, kemudian karena tipe datanya sebuah *string* maka pada *parameter* ke 3 dari **method** *bindParam* menjadi **PDO::PARAM\_STR** dan *parameter* ke 4 berisi nilai angka 15 adalah *max character* untuk *string* yang bisa ditampung.

Jika *code* diatas dieksekusi maka hasilnya :

Nama : Ayunda Faza Maudya

Email : mod@email.com

Nama : Gun Gun Febrianza

Email : gungunfebrianza@email.com

Nama : Christa Agung Winarno

**Table KRUNA 11**

Apa anda masih ingat dan faham apa itu <i>method fetch</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>numeric index</i> dan <i>string index</i> pada sebuah <i>array</i> ?	Ya / Tidak
Apa anda faham apa itu <i>prepared statement</i> ?	Ya / Tidak
Apa anda tahu manfaat <i>prepared statement</i> ?	Ya / Tidak
Apa anda tahu apa itu <i>persistent connection</i> ?	Ya / Tidak
Apa anda faham apa itu <i>method bindParam</i> ?	Ya / Tidak
Apa anda faham apa perbedaan <i>variable named placeholder</i> dan <i>question mark placeholder</i> pada <i>prepared statement</i> ?	Ya / Tidak
Apa anda faham apa itu <i>method prepare</i> ?	Ya / Tidak
Apa anda faham apa itu <i>method execute</i> ?	Ya / Tidak
Tutup halaman ini apa anda sudah yakin memahami <i>chapter 15</i> dan <i>16</i> ?	Ya / Tidak
Apa anda sudah melakukan Studi Kasus Belajar Salah dari chap <i>14</i> , <i>15</i> dan <i>16</i> ?	Ya / Tidak

## Chapter 17 : *Fetch Style*

Sebelumnya pada *chapter* 14 kita telah belajar mengenal *method Fetch* dalam sebuah *PDOstatement Object* hasil dari *method Query* pada *class PDO*. Kini kita akan mengenal lebih dalam *method fetch* dalam prepared statement, disini *method fetch* akan menggunakan *parameter* dan setiap *parameter* mempunyai style masing masing dalam memberikan *result set*. Perbedaananya kini kita akan menggunakan *method Fetch* dalam sebuah *PDOstatement Object* hasil dari *method prepare*.

Agar lebih faham buatlah sebuah file bernama OOP13\_FetchStyle.php didalam *folder oop* dan masukan *code* dibawah ini :

```
<?php
try
{
    $con = new PDO('mysql:host=localhost;dbname=oop', 'root', '',
array(PDO::ATTR_PERSISTENT => true));

} catch (PDOException $e){
    echo "Error!: " . $e->getMessage() . "<br/>";
    die();
}

/* Mengeksekusi prepared statement dengan melakukan operasi binding variable */
$id = 3;
$stmt = $con->prepare('SELECT nama, email
    FROM table1');
$stmt->bindParam(':id', $id, PDO::PARAM_INT);
$stmt->execute();
$result = $stmt->fetch(PDO::FETCH_ASSOC);
print_r($result);
echo "<br> Nama : " . $result['nama'] . "<br>";
echo "Email : " . $result['email'] . "<br> <br>";

?>
```

Pada *code* diatas kita melakukan operasi *fetching* menggunakan *method fetch* dengan *style fetch\_assoc* yang akan menghasilkan sebuah *array* yang indexnya menggunakan nama *column* dalam sebuah *result set*. Untuk melihat bentuk *result set* itu sendiri kita bisa menggunakan *statement code* selanjutnya :

```
print_r($result);
```

Kemudian jika *kode* dieksekusi hasilnya adalah sebagai berikut :



Array ( [nama] => Gun Gun Febrianza [email] => gungunfebrianza@email.com )

Ini artinya jika kita ingin mengetahui nilai atau elemen yang ada pada *column* nama dan *column* email kita tinggal mengakses Array **\$result** menggunakan *statement code* selanjutnya :

```
echo "<br> Nama : " . $result['nama'] . "<br>";  
echo "Email : " . $result['email'] . "<br> <br>";
```

Jika dieksekusi maka hasilnya :



Array ( [nama] => Gun Gun Febrianza [email] => gungunfebrianza@email.com )  
Nama : Gun Gun Febrianza  
Email : gungunfebrianza@email.com

Apa yang terjadi jika kita menambahkan *statement code* dibawah ini lagi?

```
$result = $sth->fetch(PDO::FETCH_ASSOC);  
print_r($result);  
echo "<br> Nama : " . $result['nama'] . "<br>";  
echo "Email : " . $result['email'] . "<br> <br>";
```

Maka jika dieksekusi hasilnya adalah *record* selanjutnya :



Array ( [nama] => Gun Gun Febrianza [email] => gungunfebrianza@email.com )  
Nama : Gun Gun Febrianza  
Email : gungunfebrianza@email.com

Array ( [nama] => Christa Agung Winarno [email] => CAW@email.com )  
Nama : Christa Agung Winarno  
Email : CAW@email.com

Perhatikan *record* pada table1 dibawah ini :

+ Options

		id	nama	email
<input type="checkbox"/>	Edit  Copy  Delete	1	Gun Gun Febrianza	gungunfebrianza@email.com
<input type="checkbox"/>	Edit  Copy  Delete	2	Christa Agung Winarno	CAW@email.com
<input type="checkbox"/>	Edit  Copy  Delete	3	Ayunda Faza Maudya	mod@email.com

Jika *statement code* diatas dieksekusi lagi maka hasilnya adalah *record* selanjutnya yaitu Ayunda Faza Maudya.

*Sample Full Code* ada pada *file zip* yang telah *download*. Nama *File* : OOP13\_FetchStyle.php

Selanjutnya kita akan mempelajari *fetch style* yang kedua yaitu *fetch\_BOTH*. Agar lebih faham buatlah sebuah *file* bernama OOP14\_FetchStyleBoth.php didalam *folder* oop dan masukan *code* dibawah ini :

```
<?php
try
{
    $con = new PDO('mysql:host=localhost;dbname=oop', 'root', '',
array(PDO::ATTR_PERSISTENT => true));

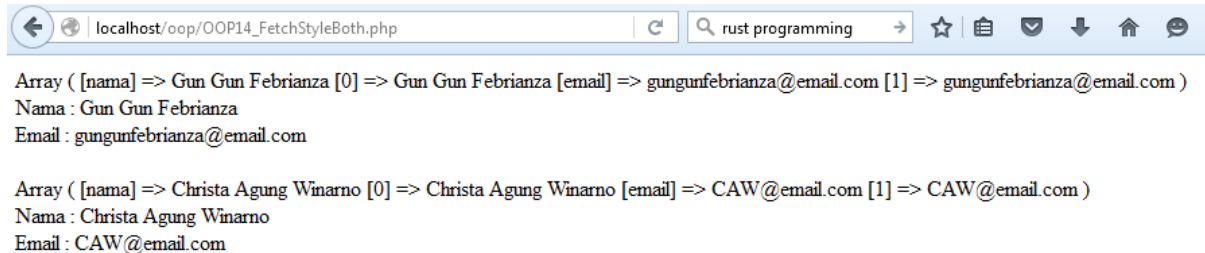
} catch (PDOException $e){
    echo "Error!: " . $e->getMessage() . "<br/>";
    die();
}

/* Mengeksekusi prepared statement dengan melakukan operasi binding variable */
$id = 3;
$stmt = $con->prepare('SELECT nama, email
    FROM table1');
$stmt->bindParam(':id', $id, PDO::PARAM_INT);
$stmt->execute();
$result = $stmt->fetch(PDO::FETCH_BOTH);
print_r($result);
echo "<br> Nama : " . $result[0] . "<br>";
echo "Email : " . $result[1] . "<br> <br>";

$result = $stmt->fetch(PDO::FETCH_BOTH);
print_r($result);
echo "<br> Nama : " . $result['nama'] . "<br>";
echo "Email : " . $result['email'] . "<br> <br>";

?>
```

Perbedaannya adalah pada *result set*, *array* yang dihasilkan kontennya lebih banyak sebab kita bisa mengakses elemen didalamnya menggunakan dua cara sekaligus yaitu by *index* (angka) atau *column* (nama) seperti yang telah kita pelajari sebelumnya dalam chapter 13. Jika kode diatas dieksekusi maka hasilnya :



```
Array ( [nama] => Gun Gun Febrianza [0] => Gun Gun Febrianza [email] => gungunfebrianza@email.com [1] => gungunfebrianza@email.com )
Nama : Gun Gun Febrianza
Email : gungunfebrianza@email.com

Array ( [nama] => Christa Agung Winarno [0] => Christa Agung Winarno [email] => CAW@email.com [1] => CAW@email.com )
Nama : Christa Agung Winarno
Email : CAW@email.com
```

Kita bisa melihat perbedaan pada result setnya.

Selanjutnya kita akan mempelajari *fetch style* yang ketiga yaitu *fetch\_LAZY*. Agar lebih faham buatlah sebuah file bernama *OOP15\_FetchStyleLazy.php* didalam *folder* *oop* dan masukan *code* dibawah ini :

```
<?php
try
{
    $con = new PDO('mysql:host=localhost;dbname=oop', 'root', '',
array(PDO::ATTR_PERSISTENT => true));

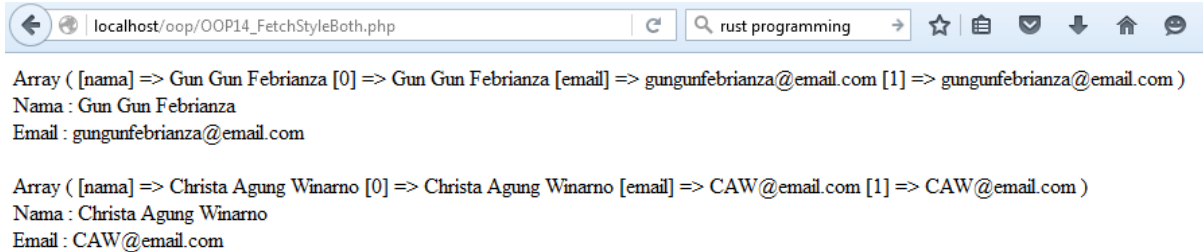
} catch (PDOException $e){
    echo "Error!: " . $e->getMessage() . "<br/>";
    die();
}

/* Mengeksekusi prepared statement dengan melakukan operasi binding variable */
$id = 3;
$stmt = $con->prepare('SELECT nama, email
FROM table1');
$stmt->bindParam(':id', $id, PDO::PARAM_INT);
$stmt->execute();
$result = $stmt->fetch(PDO::FETCH_LAZY);
print_r($result);
echo "<br> Nama : " . $result[0] . "<br>";
echo "Email : " . $result[1] . "<br> <br>";

$result = $stmt->fetch(PDO::FETCH_LAZY);
print_r($result);
echo "<br> Nama : " . $result['nama'] . "<br>";
echo "Email : " . $result['email'] . "<br> <br>";

?>
```

Perbedaannya adalah pada *result set*, yaitu sebuah *object* PDO, didalamnya terdapat sebuah *column* yang selanjutnya disebut *property* karena dia merupakan sebuah *object*. Meskipun dia adalah sebuah *object* kita tetap bisa mengaksesnya sebagai mana kita mengakses sebuah *array*. Didalamnya terdapat *column* *queryString* yang elemennya berisi SQL Statement dari mana dia diciptakan dan *column* lainnya. Jika kode diatas dieksekusi maka hasilnya :



```
Array ( [nama] => Gun Gun Febrianza [0] => Gun Gun Febrianza [email] => gungunfebrianza@email.com [1] => gungunfebrianza@email.com )
Nama : Gun Gun Febrianza
Email : gungunfebrianza@email.com

Array ( [nama] => Christa Agung Winarno [0] => Christa Agung Winarno [email] => CAW@email.com [1] => CAW@email.com )
Nama : Christa Agung Winarno
Email : CAW@email.com
```

Table KRUNA 12	
Apa anda masih faham dan hafal segala hal yang ada didalam Table KRUNA 11 (buka lagi)?	
Apa anda masih ingat dan faham apa itu <i>prepared statement</i> ?	Ya / Tidak
Apa anda masih ingat dan faham manfaat <i>prepared statement</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>persistent connection</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa itu <i>method bindParam</i> ?	Ya / Tidak
Apa anda masih ingat dan faham apa perbedaan <i>variable named placeholder</i> dan <i>question mark placeholder</i> pada <i>prepared statement</i> ?	Ya / Tidak
Apa anda masih ingat dan faham faham apa itu <i>method prepare</i> ?	Ya / Tidak
Apa anda masih ingat dan faham itu <i>method execute</i> ?	Ya / Tidak
Tutup halaman ini apa anda sudah yakin memahami <i>chapter 17</i> ?	Ya / Tidak
Apa anda sudah melakukan Studi Kasus Belajar Salah dari chap 17?	Ya / Tidak



## Chapter 18 : Finishing Touch

Akhirnya sampai juga pada *chapter* ini, sebuah *chapter* yang dinanti nanti karena kita akan mengimplementasikan semua yang telah kita pelajari. Kita akan membuat sebuah *web application* dengan *Style* OOP berpadu dengan PDO untuk melakukan operasi CRUD. Pada *chapter* ini kita juga akan mengenal ***framework Bootstrap*** yang sangat populer untuk desain ***graphic user interface*** pada *web application* yang akan kita buat. Berikut adalah tampilan *web application* yang akan kita buat :

+ Tambah Record

#	Nama	Email	Actions	
1	Gun Gun Febrianza	gungunfebrianza@email.com		
2	Christa Agung Winarno	CAW@email.com		
3	Ayunda Faza Maudya	mod@email.com		

1

2

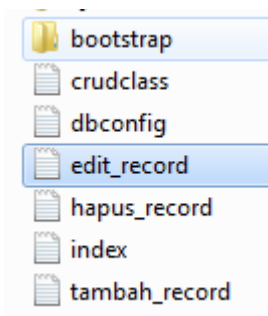
Next

Last

Selamat Belajar :) - Jika ada pertanyaan ask me [here!](#)

Fiturnya masih sederhana hanya **CREATE**, **READ**, **UPDATE** dan **DELETE**. Tapi saya percaya dari kerangka dasar ini kedepanya anda bisa membuat ***web application*** yang lebih baik sesuai dengan kebutuhan anda, terkadang orang orang kebingungan darimana mereka harus memulai untuk membuat sebuah *web application* dan mereka hanya memerlukan pelatuk yang tepat untuk memulai agar bisa melakukan eksplorasi segalanya sendiri. Semoga kajian ini menjadi pelatuk yang tepat untuk pembaca.

Sebelum kita memulainya ada beberapa hal yang harus kita persiapkan, silahkan anda *extract file* yang ada didalam **webapplication.zip** kedalam *folder* oop untuk mendapatkan seluruh *source code* yang akan kita kaji di *chapter* ini berikut adalah strukturnya:



Silahkan buka *file dbconfig.php* untuk melihat isi *code* yang ada didalamnya :

```
1  <?php
2
3  try
4  {
5      $con = new PDO('mysql:host=localhost;
6  }
7  catch(PDOException $e)
8  {
9      echo $e->getMessage();
10 }
11
12 include_once 'crudclass.php';
13 $crud = new crud($con);
14
15 ?>
```

Seperti biasa pada baris ke 5 kita akan membuat *object* **\$con** menggunakan *Class PDO*, bersamaan dengan informasi *data source name* dan *user credential* pada *parameternya* kemudian kita menggunakan *persistent connection* semua ini telah kita pelajari di *chapter* sebelumnya.

Kemudian pada baris ke 12 *file crudclass.php* dipanggil menggunakan fungsi **include\_once**. Bagi yang sudah faham **fungsi include\_once** skip saja bagian ini, bagi yang belum kita akan mempelajari perbedaan antara *include* dan *include\_once*.

Untuk memahaminya pertama buatlah 3 buah file :

1. test.php
2. Include.php
3. Include\_once.php

Isi *code* didalam *index.php* adalah sebagai berikut :

```
<?PHP
include("include.php");
include("include.php");
include("include.php");
include_once("include_once.php");
include_once("include_once.php");
include_once("include_once.php");
?>
```

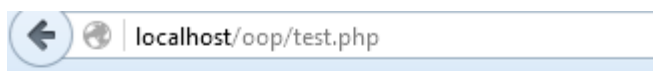
Pada *index.php* kita akan melihat perbedaan **include** dan **include\_once**, kemudian pada *file include.php* masukan *code* dibawah ini :

```
<?PHP
echo "Saya adalah mrs.mod yang ada di include.php!<br>";
?>
```

kemudian pada *file include\_once.php* masukan *code* dibawah ini :

```
<?PHP
echo "Saya adalah mr.g yang ada di include_once.php!<br>";
?>
```

Maka jika dieksekusi *script* test.php hasilnya adalah :



Saya adalah mrs.mod yang ada di include.php!  
Saya adalah mrs.mod yang ada di include.php!  
Saya adalah mrs.mod yang ada di include.php!  
Saya adalah mr.g yang ada di include\_once.php!

Perbedaanya sangat sederhana **include()** bisa memanggil *file* yang sama untuk beberapa kali dan **include\_once()** hanya akan memanggil *file* yang sama sekali saja.

Kemudian pada baris ke 13 kita membuat sebuah *object* bernama **\$crud** menggunakan *class* **crud** yang ada didalam *file* **include\_once 'crudclass.php'**; berikut dengan *parameternya* \$con.

```
$crud = new crud($con);
```

Jika kita lihat pada baris ke 3 *script* yang ada didalam **crudclass.php** maka isinya hanya satu *class* dengan nama *class* **crud** seperti pada gambar dibawah ini :

```
3  class crud
4  {
5      private $db;
6
7      function __construct($con)
8      {
9          $this->db = $con;
10     }
```

Ini artinya *object* yang bernama **\$crud** bisa mengakses seluruh *method/function* yang ada didalam *class* **crud** dan *class* **crud** itu sendiri berada didalam *file* **crudclass.php**, kemudian pada *class* **crud** kita melihat fungsi *construct* yang akan menerima *paramater object* **\$con** yang telah kita buat didalam *file* **dbconfig.php** sehingga saat sebuah *object* dibuat menggunakan *class* **crud** dia harus memasukan *parameter* sebuah *object* untuk membuat sebuah *object*. (hayoo logikanya harus main :p wkwkwkwk)

```
$crud = new crud($con);
```

Sekali lagi **\$con** adalah *object* yang kita buat didalam *file dbconfig.php* yang digunakan untuk membuat *object* **\$crud** menggunakan *class crud*.

Silahkan anda buka di dalam *crudclass.php* terdapat 7 *method/function* diantaranya adalah :

1. **buat()** untuk mengeksekusi perintah *INSERT SQL Statement*

```
public function buat($nama,$email)
{
    try
    {
        $stmt = $this->db->prepare("INSERT INTO table1(nama,email) VALUES(:nama, :email)");
        $stmt->bindParam(":nama",$nama);
        $stmt->bindParam(":email",$email);
        $stmt->execute();
        return true;
    }
    catch(PDOException $e)
    {
        echo $e->getMessage();
        return false;
    }
}
```

2. **getID()** fungsi ini akan digunakan saat kita akan melakukan *update database*

```
public function getID($id)
{
    $stmt = $this->db->prepare("SELECT * FROM table1 WHERE id=:id");
    $stmt->execute(array(":id"=>$id));
    $editRow=$stmt->fetch(PDO::FETCH_ASSOC);
    return $editRow;
}
```

3. **update()** fungsi ini akan digunakan saat kita melakukan *update database*

```
public function update($id,$nama,$email)
{
    try
    {
        $stmt=$this->db->prepare("UPDATE table1 SET nama=:nama,
                                email=:email
                                WHERE id=:id ");
        $stmt->bindParam(":nama",$nama);
        $stmt->bindParam(":email",$email);
        $stmt->bindParam(":id",$id);
        $stmt->execute();

        return true;
    }
    catch(PDOException $e)
    {
        echo $e->getMessage();
        return false;
    }
}
```

4. **hapus()** fungsi ini akan kita gunakan untuk menghapus suatu *record*

```
public function hapus($id)
{
    $stmt = $this->db->prepare("DELETE FROM table1 WHERE id=:id");
    $stmt->bindParam(":id",$id);
    $stmt->execute();
    return true;
}
```

5. **lihatdata()** fungsi ini digunakan untuk menampilkan data pada halaman utama

```
public function lihatdata($query)
{
    $stmt = $this->db->prepare($query);
    $stmt->execute();

    if($stmt->rowCount()>0)
    {
        while($row=$stmt->fetch(PDO::FETCH_ASSOC))
        {
            ?>
            <tr>
            <td><?php echo($row['id']); ?></td>
            <td><?php echo($row['nama']); ?></td>
            <td><?php echo($row['email']); ?></td>
            <td align="center">
            <a href="edit_record.php?edit_id=<?php echo($row['id']); ?>">
            <i class="glyphicon glyphicon-edit"></i></a>
            </td>
            <td align="center">
            <a href="hapus_record.php?delete_id=<?php echo($row['id']); ?>">
            <i class="glyphicon glyphicon-remove-circle"></i></a>
            </td>
            </tr>
            <?php
        }
    }
}
```

6. **paging()** fungsi ini untuk menampilkan *record* yang akan muncul dihalaman utama

```
public function paging($query,$records_per_page)
{
    $starting_position=0;
    if(isset($_GET["page_no"]))
    {
        $starting_position=( $_GET["page_no"]-1)*$records_per_page;
    }
    $query2=$query." limit $starting_position,$records_per_page";
    return $query2;
}
```

7. **paginglink()** fungsi untuk menampilkan halaman yang menampilkan *record* dihalaman utama scriptnya agak panjang :

```
public function paginglink($query,$records_per_page)
{

    $self = $_SERVER['PHP_SELF'];

    $stmt = $this->db->prepare($query);
    $stmt->execute();

    $total_no_of_records = $stmt->rowCount();
```

Sekarang kita akan mulai memahami halaman utama *web application* yang kita buat yaitu **index.php**, silahkan dibuka untuk dilihat dan dikaji kodenya disana kita akan sambil mempelajari *bootstrap*.

```
1  <?php
2  include_once 'dbconfig.php';
3  ?>
```

Pada *source code* index.php kita memanggil fungsi ***include\_once*** file ***dbconfig.php***. Sehingga dihalaman ini kita memiliki *object* **crud** yang akan kita gunakan memanggil *method/function* **lihatdata()** yang ada didalam **class crud**.

Kemudian pada baris ke 10 didalam tag <head> kita memasukan *asset bootstrap* yaitu :

```
9  <title>Finishing Touch</title>
10 <link href="bootstrap/css/bootstrap.min.css" rel="stylesheet" >
```

Sehingga kita bisa memanfaatkan sekumpulan *class* yang telah disediakan oleh **bootstrap** untuk kita gunakan. Pada kode html selanjutnya yaitu pada baris 15 sampai 19 kita membuat sebuah *tag* div yang menggunakan *class container* milik **bootstrap** untuk membungkus suatu *statement code* yang akan menjadi pemisah.:

```
15 <div class="container">
16 <a href="tambah_record.php" class="btn btn-large btn-success">
17 <i class="glyphicon glyphicon-plus"></i>
18 &nbsp;Tambah Record</a>
19 </div>
```

Kemudian kita membuat sebuah *button* menggunakan *class btn-success* milik bootstrap, jika anda ingin mengganti warnanya silahkan lihat gambar ini :



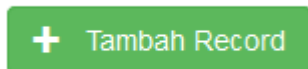
Berikut nama *class* untuk mengubahnya, pada kode diatas kita menggunakan **btn-success**

- `.btn-default`
- `.btn-primary`
- `.btn-success`
- `.btn-info`
- `.btn-warning`
- `.btn-danger`
- `.btn-link`

Selain *button* kita juga menggunakan *glyphicon* milik **bootstrap**

```
<i class="glyphicon glyphicon-plus"></i>
```

Pada kode diatas kita menggunakan *class* **glyphicon-plus** sehingga tombol *button* yang ada dihalaman *web application* yang kita buat memiliki *symbol* plus seperti gambar dibawah ini :



**Bootstrap** menyediakan 260 **glyphicons** yang bisa kita gunakan didalam sebuah teks, *button*, toolbar, *navigation*, *form* dan masih banyak lagi. Untuk lebih lengkapnya silahkan akses :

**glyphicons.com**

Disana ada banyak sekali contoh dan nama *class* yang bisa kita gunakan untuk membuat sebuah **glyphicons**, dibawah ini adalah *sample glyphicon* yang telah disediakan.



Kemudian pada kode html selanjutnya yaitu pada baris 23 sampai 45 kita membuat sebuah *table* yang didalamnya juga terdapat *script* php yang akan kita gunakan untuk menampilkan data ;

```
23 <div class="container">
24   <table class='table table-bordered table-responsive'>
25   <tr>
26     <th>#</th>
27     <th>Nama</th>
28     <th>Email</th>
29     <th colspan="2" align="center">Actions</th>
30   </tr>
31   <?php
32     $query = "SELECT * FROM table1";
33     $records_per_page=3;
34     $newquery = $crud->paging($query,$records_per_page);
35     $crud->lihatdata($newquery);
36   ?>
37   <tr>
38     <td colspan="7" align="center">
39       <div class="pagination-wrap">
40         <?php $crud->paginglink($query,$records_per_page); ?>
41       </div>
42     </td>
43   </tr>
44 </table>
45
```

Pada kode diatas tepatnya pada baris ke 24 kita membuat *table* menggunakan *class* milik *bootstrap*. Ada dua *class* yang digunakan sekaligus yaitu **class table-bordered dan table-responsive**, efek dari *class table-bordered* adalah *table* memiliki *border* disemua sisi *table* dan *cell*. Anda bisa menggantinya menggunakan *class*

1. *table-striped* memberikan efek zebra pada *table* (setiap baris ada perbedaan warna)
2. *table-hover* memberikan efek hover pada *table* (perubahan warna saat cursor menyentuh *table*)

**Table-responsive** artinya *table* akan secara otomatis menyesuaikan ukuranya sesuai dengan ukuran *mobile*, *tablet* atau *desktop*. Selanjutnya kita akan mengkaji kode php yang kita gunakan untuk menampilkan sekumpulan *record* dalam *database* dihalaman utama *web application* yaitu pada baris 31 sampai 36.

```
31 <?php
32   $query = "SELECT * FROM table1";
33   $records_per_page=3;
34   $newquery = $crud->paging($query,$records_per_page);
35   $crud->lihatdata($newquery);
36 ?>
```



Pada *code* diatas baris 32 kita membuat sebuah *variable* bernama **\$query** untuk menampung sebuah **SQL Statement SELECT** dan pada baris ke 33 *variable* **\$records\_per\_page** untuk membatasi jumlah *record* yang akan ditampilkan dihalaman utama, karena sebelumnya di halaman ini kita sudah memanggil *file dbconfig.php* menggunakan fungsi *include\_once* maka kita memiliki *object* **\$crud**.

Selanjutnya pada baris 34 dengan *object* **\$crud** kita memanggil fungsi *paging* *parameternya* adalah *variable* **\$query** dan **\$records\_per\_page** yang kemudian hasilnya disimpan didalam *variable* **\$newquery**. Dibawah ini adalah fungsi *paging* :

```
104 public function paging($query,$records_per_page)
105 {
106     $starting_position=0;
107     if(isset($_GET["page_no"]))
108     {
109         $starting_position=( $_GET["page_no"]-1)*$records_per_page;
110     }
111     $query2=$query." limit $starting_position,$records_per_page";
112     return $query2;
113 }
```

Saat pertama kali halaman utama dibuka nilai pada *variable* **\$starting\_position=0** selanjutnya *SQL Statement Select* yang tersimpan didalam *variable* **\$query** akan menerima *parameter SQL Statement* baru yaitu perintah **limit** untuk membatasi dengan nilai **\$starting\_position = 0** dan **\$records\_per\_page = 3**, maka isi dari *variable* **\$query2** pada baris ke 111 adalah **SELECT \* FROM table1 limit 0,3**.

Isi nilai dari *Variable* **\$query2** kemudian disimpan kedalam *variable* **\$newquery** dihalaman *index.php* yaitu pada baris ke 34.

```
31 <?php
32     $query = "SELECT * FROM table1";
33     $records_per_page=3;
34     $newquery = $crud->paging($query,$records_per_page);
35     $crud->lihatdata($newquery);
36     ?>
```

Kemudian pada baris ke 35 *object* **\$crud** mengeksekusi *method/function* **lihatdata()** dengan *variable* **\$newquery** sebagai *parameternya*. Dibawah ini adalah fungsi *lihatdata()* :

```

67 public function lihatdata($query)
68 {
69     $stmt = $this->db->prepare($query);
70     $stmt->execute();
71
72     if($stmt->rowCount()>0)
73     {
74         while($row=$stmt->fetch(PDO::FETCH_ASSOC))
75         {
76             <?>
77             <tr>
78                 <td><?php echo($row['id']); ?></td>
79                 <td><?php echo($row['nama']); ?></td>
80                 <td><?php echo($row['email']); ?></td>
81                 <td align="center">
82                     <a href="edit_record.php?edit_id=<?php echo($row['id']); ?>">
83                         <i class="glyphicon glyphicon-edit"></i></a>
84                     </td>
85                 <td align="center">
86                     <a href="hapus_record.php?delete_id=<?php echo($row['id']); ?>">
87                         <i class="glyphicon glyphicon-remove-circle"></i></a>
88                     </td>
89                 </tr>
90             <?php
91         }
92     }

```

Pada fungsi lihatdata tepatnya pada baris 69 dan 70 kita melihat *statement code* dibawah ini :

```

$stmt = $this->db->prepare($query);
$stmt->execute();

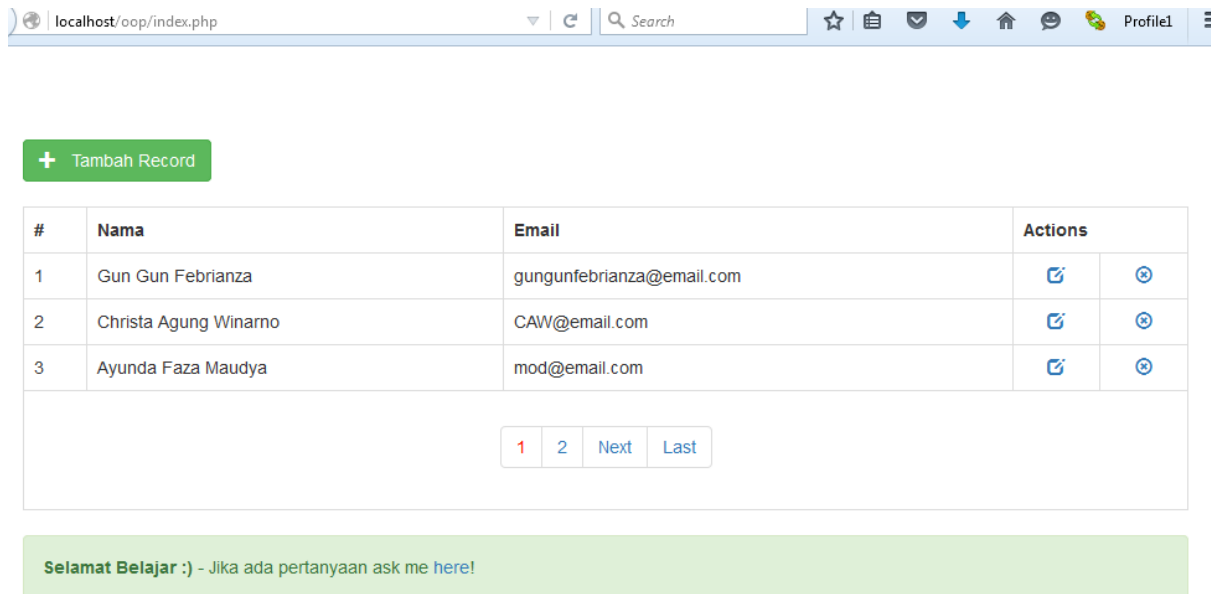
```

**db** adalah *object* \$con yang telah kita buat sebelumnya didalam *file dbconfig.php*, nilai *variable* \$newquery pada *parameter* fungsi **lihatdata()** akan dilemparkan kedalam *variable* \$query di baris ke 69 yang kemudian pada baris ke 70 akan dieksekusi menggunakan *method execute*.







**rowCount** pada baris 72 adalah sebuah *method* untuk mendapatkan angka jumlah baris yang berhasil dieksekusi oleh sebuah *SQL Statement* sebelumnya, sebelumnya kita mengeksekusi *SQL Statement* yang ada didalam *variable* \$query yaitu `SELECT * FROM table1 limit 0,3` hasilnya adalah 3 baris *record*.

Karena nilainya diatas 0 maka *statement code* selanjutnya dieksekusi. Pada baris 73 kita menggunakan perulangan *while* untuk melakukan operasi *fetching* menggunakan *method fetch* dengan *style fetch\_assoc* yang telah kita pelajari sebelumnya. Hasilnya disimpan kedalam *array* \$row, kemudian kita bisa mengakses elemen yang ada didalamnya menggunakan nama kolomnya seperti pada baris kode 78 - 80. Hasilnya akan disimpan kedalam *table* di halaman utama *web application* yang kita buat yaitu **index.php**.

Sehingga jika kita mengakses halaman utama *web application* yang kita buat kita sudah bisa melihat *record* tersusun rapi secara otomatis.



+ Tambah Record

#	Nama	Email	Actions	
1	Gun Gun Febrianza	gungunfebrianza@email.com		
2	Christa Agung Winarno	CAW@email.com		
3	Ayunda Faza Maudya	mod@email.com		

1 2 Next Last

Selamat Belajar :) - Jika ada pertanyaan ask me [here!](#)

Kemudian pada index.php selanjutnya kita lihat *statement code* pada baris ke 40 :

```
37 <tr>
38 <td colspan="7" align="center">
39 <div class="pagination-wrap">
40 <?php $crud->paginglink($query,$records_per_page); ?>
41 </div>
42 </td>
```

Object **\$crud** mengeksekusi *method/function* **paginglink()** dengan *parameter variable* **\$query** dan **\$records\_per\_page** untuk melakukan *pagination*, sehingga persatu halaman hanya menampilkan 3 *record* saja jika terdapat lebih dari 3 *record* maka akan dimasukan kehalaman dua dan seterusnya. Dibawah ini adalah fungsi **paginglink** :

```
115 public function paginglink($query,$records_per_page)
116 {
117
118     $self = $_SERVER['PHP_SELF'];
119
120     $stmt = $this->db->prepare($query);
121     $stmt->execute();
122
123     $total_no_of_records = $stmt->rowCount();
124
125     if($total_no_of_records > 0)
126     {
127         ?><ul class="pagination"><?php
128         $total_no_of_pages=ceil($total_no_of_records/$records_per_page);
```

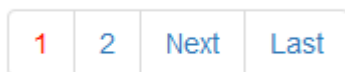
Pada baris ke 119 *Variable* **\$self** menyimpan nilai sebuah *super global variable*. **\$\_SERVER** adalah *super global variable* yang digunakan untuk menyimpan informasi *headers*, *path*, dan lokasi *script*. **\$\_SERVER['PHP\_SELF']**; akan menyimpan alamat dan nama *file* dari *script* saat ini yang sedang dieksekusi, jika *script* ini dieksekusi di halaman ini <http://localhost/oop/index.php> maka akan menghasilkan nilai **/oop/index.php**.

Pada *statement code* selanjutnya pada baris 120 sampai 21 **SQL Statement SELECT \* FROM table1** dieksekusi dan *variable* **\$total\_no\_of\_records** pada baris ke 123 memiliki nilai 4 dikarenakan hasil eksekusi dari perintah *SQL statement* sebelumnya menghasilkan 4 baris *record*.

Kemudian pada *statement code* selanjutnya pada baris ke 123 *variable* **\$total\_no\_of\_pages** menyimpan hasil dari pembagian antara *variable* **\$total\_no\_of\_records = 4** dan *variable* **\$record\_per\_page = 3**.

```
?><ul class="pagination"><?php
$total_no_of_pages=ceil($total_no_of_records/$records_per_page);
$current_page=1;
```

Hasil dari  $4/3 = 1,3$  tetapi karena dia menggunakan *method/function* **ceil()** maka hasilnya akan dibulatkan menjadi 2 sehingga pada halaman utama *web application* kita akan tampil page 1 dan page 2.



Sekarang silahkan anda buka *file* **tambah\_record.php** untuk melihat isi *code* yang ada didalamnya :

```
2   include_once 'dbconfig.php';
3   if(isset($_POST['btn-save']))
4   {
5       $nama = $_POST['nama'];
6       $email = $_POST['email'];
7
8       if($crud->buat($nama,$email))
9       {
10          header("Location: tambah_record.php?inserted");
11      }
```

Pada baris ke 2 Seperti biasa kita akan memanggil *file* **dbconfig.php** menggunakan fungsi **include\_once** agar bisa menggunakan *Object* **\$crud**, sebab selanjutnya *Object* **\$crud** akan memanggil *method/function* **buat()** seperti pada baris ke 8.

Data yang kita masukan didalam *textbox* halaman *tambah\_record.php* akan tersimpan didalam *variable* `$_POST` yang selanjutnya disimpan kedalam *variable* `$nama` dan `$email` yaitu pada baris 5 & 6.

Nama	<input type="text" value="Kaiz"/>
Email	<input type="text" value="kaizera@gmail.com"/>
<input type="button" value="+ Tambah Record"/> <input type="button" value="◀ Kembali ke halaman utama"/>	

Selamat Belajar :) - Jika ada pertanyaan ask me [here!](#)

*Variable* `$nama` dan `$email` akan menjadi *parameter* dalam *method/function* `buat()` dan diakses oleh *Object* `$crud` yang selanjutnya akan diolah oleh fungsi `buat()` :

```
12 public function buat($nama,$email)
13 {
14     try
15     {
16         $stmt = $this->db->prepare("INSERT INTO table1(nama,email) VALUES(:nama, :email)");
17         $stmt->bindParam(":nama",$nama);
18         $stmt->bindParam(":email",$email);
19         $stmt->execute();
20         return true;
21     }
```

Pada baris ke 16 *Variable* `$nama` dan `$email` akan di binding menggunakan *method/function* `bindParam` agar bisa tersimpan kedalam parameter named placeholder yang ada di **SQL Statement INSERT**. Jika berhasil dieksekusi maka akan menghasilkan nilai *return* TRUE. Jika hasilnya TRUE maka *statement code* pada **tambah\_record.php** pada baris ke 10 akan dieksekusi :

```
2 include_once 'dbconfig.php';
3 if(isset($_POST['btn-save']))
4 {
5     $nama = $_POST['nama'];
6     $email = $_POST['email'];
7
8     if($crud->buat($nama,$email))
9     {
10         header("Location: tambah_record.php?inserted");
11     }
```

Kemudian karena berhasil maka *statement code* pada **tambah\_record.php** yang ada pada baris 30 sampai 39 akan dieksekusi :

```

30  if(isset($_GET['inserted']))
31  {
32      ?>
33      <div class="container">
34          <div class="alert alert-info">
35              <strong>Selamat!</strong> Record telah sukses tersimpan <a href="index.php">HOME</a>!
36          </div>
37      </div>
38      <?php
39  }

```

Hasilnya adalah sebagai berikut :

**Selamat!** Record telah sukses tersimpan HOME!

Nama	<input type="text"/>
Email	<input type="text"/>

+ Tambah Record
◀ Kembali ke halaman utama

**Selamat Belajar :) -** Jika ada pertanyaan ask me [here!](#)

Kemudian jika kita cek table1 pada *database* maka akan muncul *record* baru :

+ Options

<div><div>←</div><div>→</div></div>		id	nama	email
<div><div><div></div></div><div><div></div><div></div></div><div>Edit</div><div><div></div><div></div></div><div>Copy</div><div><div></div><div></div></div><div>Delete</div></div>		1	Gun Gun Febrianza	gungunfebrianza@email.com
<div><div><div></div></div><div><div></div><div></div></div><div>Edit</div><div><div></div><div></div></div><div>Copy</div><div><div></div><div></div></div><div>Delete</div></div>		2	Christa Agung Winarno	CAW@email.com
<div><div><div></div></div><div><div></div><div></div></div><div>Edit</div><div><div></div><div></div></div><div>Copy</div><div><div></div><div></div></div><div>Delete</div></div>		3	Ayunda Faza Maudya	mod@email.com
<div><div><div></div></div><div><div></div><div></div></div><div>Edit</div><div><div></div><div></div></div><div>Copy</div><div><div></div><div></div></div><div>Delete</div></div>		4	mod	mod@gmail.com
<div><div><div></div></div><div><div></div><div></div></div><div>Edit</div><div><div></div><div></div></div><div>Copy</div><div><div></div><div></div></div><div>Delete</div></div>		5	Kaiz	kaizera@gmail.com

Begitu juga jika kita mengakses halaman utama pada *page 2* kita bisa melihat *record* baru :

+ Tambah Record

#	Nama	Email	Actions	
4	mod	mod@gmail.com		
5	Kaiz	kaizera@gmail.com		

First

Previous

1

2

Pada **column Actions** terdapat dua operasi yang bisa kita lakukan yaitu *edit* dan *delete*, Sekarang bukalah *file hapus\_record.php* untuk melihat isi *code* yang ada didalamnya.

Saat kita melakukan operasi penghapusan suatu *record*, misalkan pada kasus ini kita akan menghapus *record* nomor 5 maka kita akan dibawa kehalaman **hapus\_record.php**, *statement code* yang akan dieksekusi pertama kali adalah *code* dibawah ini yaitu pada baris 35 sampai 43 :

```

35     else
36     {
37         ?>
38         <div class="alert alert-danger">
39             <strong>Perhatian</strong> apa anda yakin akan menghapusnya ?
40         </div>
41         <?php
42     }
43     ?>
44 </div>

```

Sehingga muncul *alert* seperti yang ada dibawah ini :

**Perhatian** apa anda yakin akan menghapusnya ?

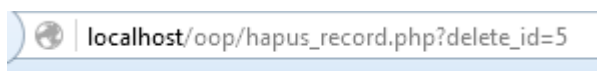
Selanjutnya *statement code* dibawah ini yang akan dieksekusi :

```

49     if(isset($_GET['delete_id']))
50     {
51         ?>
52         <table class='table table-bordered'>
53         <tr>
54             <th>#</th>
55             <th>nama</th>
56             <th>Email</th>
57         </tr>
58         <?php
59         $stmt = $con->prepare("SELECT * FROM table1 WHERE id=:id");
60         $stmt->execute(array(":id"=>$_GET['delete_id']));
61         while($row=$stmt->fetch(PDO::FETCH_BOTH))
62         {
63             ?>
64             <tr>
65                 <td><?php echo($row['id']); ?></td>
66                 <td><?php echo($row['nama']); ?></td>
67                 <td><?php echo($row['email']); ?></td>
68             </tr>
69             <?php
70         }
71         ?>
72     </table>
73     <?php
74 }

```

Pada baris ke 49 Dengan *variable* `$_GET` kita akan membaca nilai pada sebuah *array* yang ada pada *URL Parameter*. Pada kasus ini nilai dari *column name* **delete\_id** adalah **5**.



Artinya angka 5 akan disimpan kedalam *SQL Statement Select* dan dieksekusi menggunakan *method execute* operasi ini dilakukan pada baris 59 dan 60. Kemudian pada baris 61 operasi *fetching* satu baris dilakukan menggunakan *method fetch* dengan *style fetch\_assoc* untuk mendapatkan satu baris *record* dan hasilnya adalah *record* dibawah ini :

#	nama	Email
5	Kaiz	kaizera@gmail.com



Selanjutnya jika tombol yes ditekan maka *statement code* dibawah ini akan dieksekusi :



```

81 if(isset($_GET['delete_id']))
82 {
83     ?>
84     <form method="post">
85     <input type="hidden" name="id" value="<?php echo $row['id']; ?>" />
86     <button class="btn btn-large btn-primary" type="submit" name="btn-del">
87     <i class="glyphicon glyphicon-trash"></i> &nbsp; YES</button>
88     <a href="index.php" class="btn btn-large btn-success">
89     <i class="glyphicon glyphicon-backward"></i> &nbsp; NO</a>
90     </form>
91     <?php
92 }

```

Operasi *method post* dilakukan dengan mengirim nilai *hidden* angka 5 (hasil dari \$row['id']) pada baris 85 yang didapatkan dari *array \$row* dengan nilai **\$\_POST = btn-del** pada baris ke 86.

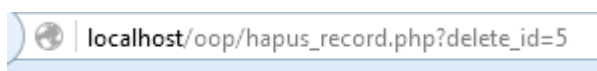
Proses selanjutnya masih terjadi dalam **hapus\_record.php**, **\$\_POST** akan terisi dengan nilai **btn-del** sehingga *statement code* dibawah ini akan dieksekusi :

```

2 include_once 'dbconfig.php';
3
4 if(isset($_POST['btn-del']))
5 {
6     $id = $_GET['delete_id'];
7     $crud->hapus($id);
8     header("Location: hapus_record.php?deleted");
9 }

```

Pada baris ke 6 \$id akan menyimpan nilai pada sebuah *array* yang ada pada **URL Parameter** yaitu angka 5.



Kemudian pada baris ke 7 **object \$crud** memanggil *method/function* hapus() dan *variable* \$id menjadi parameternya artinya kita akan menghapus *record* dengan nilai id 5. Dibawah ini adalah fungsi hapus yang ada didalam **crudclass.php**

```

59 public function hapus($id)
60 {
61     $stmt = $this->db->prepare("DELETE FROM table1 WHERE id=:id");
62     $stmt->bindParam(":id", $id);
63     $stmt->execute();
64     return true;
65 }

```

Pada baris ke 61 Bisa kita lihat pada perintah *SQL Statement* yang ada didalam *method prepare()*, DELETE FROM table1 where id=:id, ketika *SQL statement* berhasil dieksekusi ini

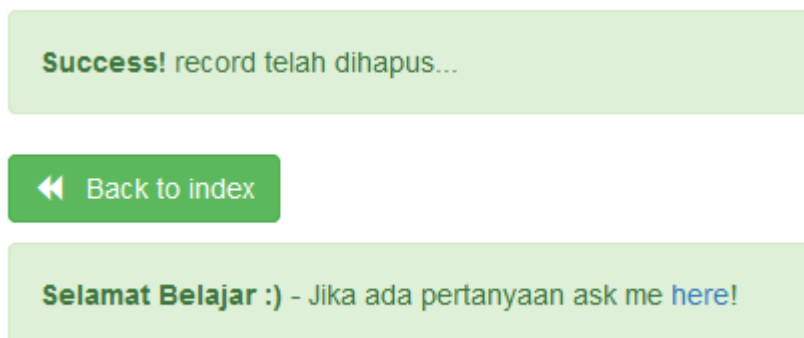
akan memberikan sebuah *return true*. Selanjutnya fungsi *header* pada baris ke 8 dalam **hapus\_record.php** akan dieksekusi :

```
2  include_once 'dbconfig.php';
3
4  if(isset($_POST['btn-del']))
5  {
6      $id = $_GET['delete_id'];
7      $crud->hapus($id);
8      header("Location: hapus_record.php?deleted");
9  }
```


Kemudian pada baris 27 *variable* `$_GET` dalam **hapus\_record.php**, akan terisi nilai **deleted** sehingga *statement code* dibawah ini akan dieksekusi :

```
26  <?php
27  if(isset($_GET['deleted']))
28  {
29      ?>
30      <div class="alert alert-success">
31          <strong>Success!</strong> record telah dihapus...
32      </div>
33      <?php
34  }
```

Maka halaman yang muncul adalah sebagai berikut :



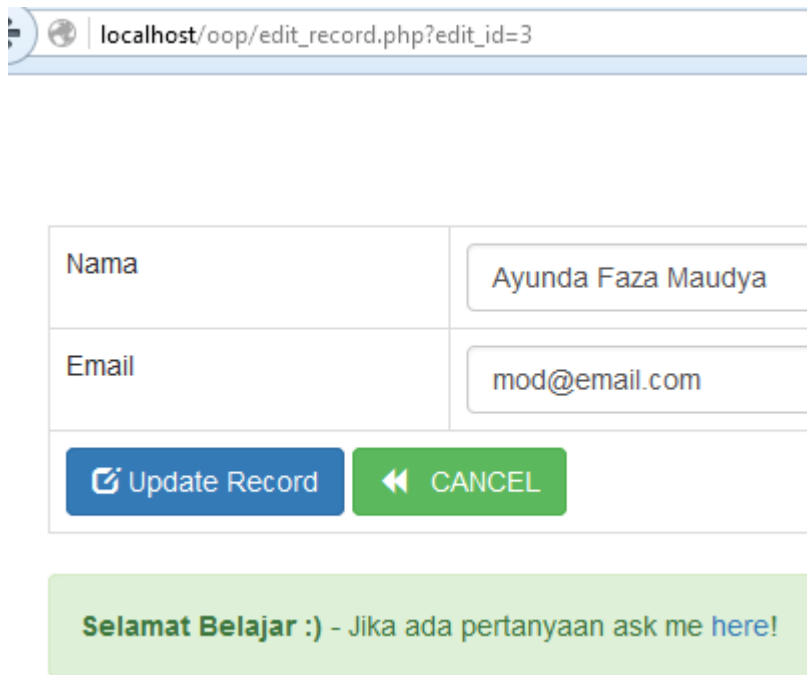
Sekarang jika kita melakukan operasi *edit record* kita coba ubah *record* ke 3 :

3	Ayunda Faza Maudya	mod@email.com	
---	--------------------	---------------	---

Maka kita akan dibawa kehalaman yang baru yaitu :

[http://localhost/oop/edit\\_record.php?edit\\_id=3](http://localhost/oop/edit_record.php?edit_id=3)

berikut tampilan halaman untuk melakukan *edit record* :



localhost/oop/edit\_record.php?edit\_id=3

Nama	Ayunda Faza Maudya
Email	mod@email.com

[Update Record](#) [CANCEL](#)

Selamat Belajar :) - Jika ada pertanyaan ask me [here!](#)

Untuk mempelajarinya silahkan buka *file edit\_record.php* untuk melihat isi *code* yang ada didalamnya. Sistematisnya saat halaman ini diakses *statement code* dibawah ini akan dieksekusi terlebih dahulu :

```
23 if(isset($_GET['edit_id']))
24 {
25     $id = $_GET['edit_id'];
26     extract($crud->getID($id));
27 }
```

Sebab *variable* **\$\_GET** akan terisi nilai dari *array* yang ada pada **URL Parameter** ini : `http://localhost/oop/edit_record.php?edit_id=3`, sehingga nilai dari *variable* **\$id** pada baris 25 adalah 3.

Kemudian pada baris ke 26 fungsi **extract()** digunakan dengan parameternya adalah hasil dari *method/function* **getID** dengan parameter **\$id = 3** yang dipanggil oleh *object* **\$crud**. Dibawah ini adalah *method/function* **getID** yang ada didalam **crudclass.php**

```
30 public function getID($id)
31 {
32     $stmt = $this->db->prepare("SELECT * FROM table1 WHERE id=:id");
33     $stmt->execute(array(":id"=>$id));
34     $editRow=$stmt->fetch(PDO::FETCH_ASSOC);
35     return $editRow;
36 }
```

Nilai dari *Variable* **\$id** akan dimasukan kedalam *named placeholder* `:id` yang ada di *SQL Statement Template* didalam *method* *prepare*, kemudian pada baris 33 eksekusi dilakukan

untuk mengirimkan nilai yang ada pada parameter *named placeholder* ke *database server*. Untuk melihat hasilnya pada baris 34 kita menggunakan operasi *fetching* dengan *fetch style fetch\_assoc* dan menyimpan hasilnya berupa *array* yang indexnya menggunakan nama *column* dalam sebuah *result set* kedalam *variable* \$editRow.

Hasil dari *variable* \$editRow akan diterima oleh fungsi *extract()* pada baris ke 26 dihalaman *edit\_record.php*.

```
23 if(isset($_GET['edit_id']))
24 {
25     $id = $_GET['edit_id'];
26     extract($crud->getID($id));
27 }
```

Selanjutnya kita bisa memanggil nilai yang ada didalamnya untuk disimpan kedalam *textbox* sebagai bahan untuk melakukan operasi *edit record*. Cara memanggilnya sangat sederhana karena hasil dari **fungsi extract()** adalah *array* maka kita bisa mengakses elemen menggunakan nama *column* hanya saja cara memanggilnya adalah **echo \$nama** dan **echo \$email** artinya kita mengakses nilai/elemen yang ada didalam *column* nama dan email.

```
58 <td>Nama</td>
59 <td><input type='text' name='first_name' class='form-control' value="<?php echo $nama; ?>"
60 </td></tr>
61
62 <tr>
63 <td>Email</td>
64 <td><input type='text' name='last_name' class='form-control' value="<?php echo $email; ?>"
65 </td></tr>
```



Sehingga pada halaman **edit\_record.php** akan muncul nilai/elemen yang kita panggil untuk diedit. Yaitu nilai dari `echo $nama` adalah Ayunda Faza Maudya dan nilai dari `$email` adalah mod@gmail.com

Nama	<input type="text" value="Ayunda Faza Maudya"/>
Email	<input type="text" value="mod@email.com"/>
<input type="button" value="Update Record"/> <input type="button" value="CANCEL"/>	

Selamat Belajar :) - Jika ada pertanyaan ask me [here!](#)


Pada kasus ini kita akan mengubah emailnya menjadi maudyayunda@gmail.com kemudian kita tekan tombol **update record**. Jika berhasil maka akan muncul halaman dibawah ini :

**Selamat** Record telah berhasil diubah :) HOME!

Nama	Ayunda Faza Maudya
Email	maudyayunda@email.com
<div><div> Update Record</div><div> CANCEL</div></div>	

**Selamat Belajar :) -** Jika ada pertanyaan ask me [here!](#)

Kemudian jika kita kembali ke halaman utama *web application* kita akan melihat email sudah berubah :

 Tambah Record

#	Nama	Email
1	Gun Gun Febrianza	gungunfebrianza@email.com
2	Christa Agung Winarno	CAW@email.com
3	Ayunda Faza Maudya	maudyayunda@email.com

Secara teknis codingnya saat kita menekan tombol *update record* method post akan dilakukan dengan mengirim nilai btn-update

```
68 <td colspan="2">
69 <button type="submit" class="btn btn-primary" name="btn-update">
70 <span class="glyphicon glyphicon-edit"></span> Update Record
```

Masih dalam *script edit\_record.php*, Variable `$_POST` akan terisi nilai btn-update sehingga *statement code* berikutnya akan dieksekusi \$id akan berisi nilai 3, \$nama dan \$email berisi nilai yang kita ganti didalam textbox.

```

3   if(isset($_POST['btn-update']))
4   {
5       $id = $_GET['edit_id'];
6       $nama = $_POST['nama'];
7       $email = $_POST['email'];
8
9       if($crud->update($id, $nama, $email))
10      {
11          $msg = "<div class='alert alert-info'>
12              <strong>Selamat</strong> Record telah
13              </div>";
14      }

```

Selanjutnya pada baris ke 9 object **\$crud** memanggil method/function update() dan menerima 3 parameter yang ada pada baris ke 5 sampai ke 7. Dibawah ini adalah fungsi **update** yang ada didalam **crudclass.php** :

```

38   public function update($id,$nama,$email)
39   {
40       try
41       {
42           $stmt=$this->db->prepare("UPDATE table1 SET nama=:nama,
43                                   email=:email
44                                   WHERE id=:id ");
45           $stmt->bindParam(":id",$id);
46           $stmt->bindParam(":nama",$nama);
47           $stmt->bindParam(":email",$email);
48           $stmt->execute();
49
50       return true;

```

Karena menghasilkan nilai *return TRUE* maka kode pada baris ke 11 di halaman **edit\_record.php** dieksekusi. *Variable \$msg* akan berisi sebuah nilai string (kode html) ucapan selamat bahwa record telah berhasil dilakukan. Sehingga pesan akan muncul karena pada baris ke 46 *statement code* akan dieksekusi.

```

43   <?php
44   if(isset($msg))
45   {
46       echo $msg;
47   }
48   ?>

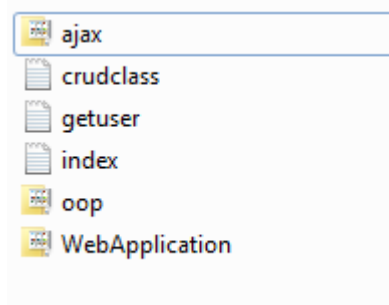
```

Selesai sudah => anda tinggal memahaminya lebih dalam lagi, mempelajarinya, memodifikasinya dan mengembangkannya lebih baik lagi. Selamat jika anda berhasil memahaminya semoga bermanfaat.

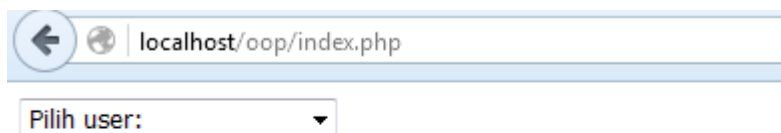
## Special Chapter : Quadruple Attack AJAX, PHP & MySQL

Pada chapter ini kita akan sedikit berkenalan dengan teknologi ajax dalam dunia web, dengan ajax kita bisa mengubah konten web tanpa harus melakukan *reload*. AJAX sendiri adalah singkatan dari Asynchronous Javascript and XML. Kita akan mengkajinya dengan singkat disini sebab versi intensif dan komprehensifnya akan saya rilis di ebook selanjutnya yang berjudul Tinju Cepat AJAX dan PHP (klo ga nemu judul lain pasti judulnya ini =)).

Langsung saja silahkan anda *extract file* ajax.rar di C:\xampp\htdocs\oop maka akan muncul tiga buah *file* php :

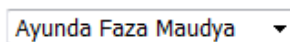


Sekarang kita coba akses <http://localhost/oop/index.php> maka akan muncul halaman seperti dibawah ini :



**Informasi mengenai user dari table1 akan tampil disini ..**

Jika kita memilih user melalui *combobox* yang ada disana maka seluruh informasi mengenai user tersebut akan tampil dibawahnya menggunakan SQL Statement SELECT tanpa *reload page* sama sekali.



Nama	Email
Ayunda Faza Maudya	maudyayunda@email.com

Sekarang kita bedah *file* index.php

```

29 <form>
30 <select name="users" onchange="showUser(this.value)">
31   <option value="">Pilih user:</option>
32   <option value="1">Gun Gun Febrianza</option>
33   <option value="2">Christa Agung Winarno</option>
34   <option value="3">Ayunda Faza Maudya</option>
35 </select>
36 </form>

```

Saat pertama kali halaman index.php diakses maka kita akan melihat *combobox* yang didalamnya hanya terdapat 3 item atau 3 user. Jika kita menekan *combobox* tersebut kemudian nilai *string* dalam *combobox* tersebut berubah maka **event onchange** pada baris ke 30 akan mengeksekusi fungsi *javascript showUser()* dan **this.value** adalah parameternya yang isinya adalah *item* yang kita pilih pada *combobox*.

Kemudian fungsi *showUser javascript* akan dieksekusi karena *parameter str* pada baris ke 4 berisi nilai kosong ma hasil dari *conditional logic* menggunakan if ini adalah **false** sehingga yang dieksekusi adalah kode pada baris ke 9 sampai 22.

```

4 function showUser(str) {
5   if (str == "") {
6     document.getElementById("txtHint").innerHTML = "";
7     return;
8   } else {
9     if (window.XMLHttpRequest) {
10      // code for IE7+, Firefox, Chrome, Opera, Safari
11      xmlhttp = new XMLHttpRequest();
12    } else {
13      // code for IE6, IE5
14      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
15    }
16    xmlhttp.onreadystatechange = function() {
17      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
18        document.getElementById("txtHint").innerHTML = xmlhttp.responseText;
19      }
20    };
21    xmlhttp.open("GET", "getuser.php?q="+str, true);
22    xmlhttp.send();
23  }
24 }

```

Jika kita menggunakan *firefox* atau *chrome* maka kode pada baris ke 11 akan dieksekusi yaitu kita membuat sebuah *object* bernama *xmlhttp*. Kemudian baris ke 16 fungsi yang akan kita gunakan untuk menerima respon dari *request* yang kita lakukan.

Pada baris ke 17 jika **xmlhttp.readyState** bernilai 4 (request selesai dan respon siap) dan **xmlhttp.status** bernilai 200 (OK) maka kita akan memanipulasi sebuah elemen yaitu **txtHint** yang ada pada kode baris ke 38. Elemen tersebut akan di isi oleh *data string* yang berasal dari **xmlhttp.responseText**.



```

29 <form>
30 <select name="users" onchange="showUser(this.value)">
31   <option value="">Pilih user:</option>
32   <option value="1">Gun Gun Febrianza</option>
33   <option value="2">Christa Agung Winarno</option>
34   <option value="3">Ayunda Faza Maudya</option>
35 </select>
36 </form>
37 <br>
38 <div id="txtHint"><b>Informasi mengenai user dari tabel akan tampil disini .. </b></div>

```

Selanjutnya pada baris 21 *object xmlhttp* memanggil *method open* yang didalamnya terdapat 3 *parameter*, pada *parameter* pertama *request* menggunakan *method* GET.

Pada *parameter* kedua berisi lokasi dari *script* PHP di *server* yang akan kita gunakan untuk menampilkan data pada elemen dengan id *txthint* dan *variable* *str* yang berisi nilai 1 – 3 tergantung dari *item* yang kita pilih pada *combobox*.

```

20   };
21   xmlhttp.open("GET","getuser.php?q="+str,true);
22   xmlhttp.send();
23   }

```

Pada *parameter* ketiga isinya ***Boolean logic TRUE***, artinya *request* akan *asynchronously*. Selanjutnya kode pada baris 22 dieksekusi yaitu *method send()* dieksekusi untuk mengirim *request* ke *server*.

Sekarang kita akan membedah *file getuser.php* yang menjadi *output* pada elemen ***txthint*** di halaman ***index.php*** sehingga memiliki table beserta informasi nama dan email dari user yang kita pilih :

```

5   table {
6     width: 100%;
7     border-collapse: collapse;
8   }
9
10  table, td, th {
11    border: 1px solid black;
12    padding: 5px;
13  }
14
15  th {text-align: left;}
16 </style>
17 </head>
18 <body>

```

Pada baris kode dari 5 sampai 15 adalah code css yang digunakan untuk mendesain sebuah table sederhana untuk menampung nilai dari operasi *SQL Statemen Select* yang akan kita lakukan nanti.

```

21 include('crudclass.php');
22 $q = intval($_GET['q']);
23 $result = $con->query("SELECT * FROM table1 WHERE id = '". $q. "'");
24
25 echo "<table>
26 <tr>
27 <th>Nama</th>
28 <th>Email</th>
29 </tr>";
30 foreach ($result as $res) :
31     echo "<tr>";
32     echo "<td>" . $res['nama'] . "</td>";
33     echo "<td>" . $res['email'] . "</td>";
34     echo "</tr>";
35 endforeach;
36 echo "</table>";
37 $con = null;

```

Pada baris 22 *variable* \$q menerima nilai **integer** dari *variable* \$\_GET karena menggunakan *method* **intval()** hasil dari *variable* \$q akan menjadi *parameter* untuk **SQL Statement Select** didalam *method* *query* yang dipanggil oleh *object* \$con.

Kemudian pada baris 25 sampai 36 nilai dari result set **SQL Statement select** didapatkan menggunakan perulangan *foreach*, *array* didalam *variable* \$res bisa kita panggil menggunakan nama kolomnya seperti yang terjadi pada baris 32 dan 33.

Terakhir pada baris 37 *object* \$con menjadi null artinya koneksi ditutup. Begitulah cara kerja sederhana sebuah Teknologi AJAX. Penasaran? Tunggu ebook selanjutnya ☺