

PDO

Sebagai alternatif dari **mysqli**, PHP menyediakan **PDO** untuk mengakses database. Selain lebih fleksibel (karena bisa dipakai untuk mengakses berbagai jenis aplikasi database), PDO juga menyediakan beberapa fitur yang tidak tersedia di **mysqli**.

Pengertian PDO

PDO (singkatan dari **PHP Data Object**), adalah object yang berfungsi untuk berkomunikasi dengan database. PDO ini mirip seperti **mysqli** yang baru saja kita bahas, akan tetapi PDO bersifat universal, yakni bisa dipakai untuk mengakses berbagai aplikasi database mulai dari MySQL / MariaDB, SQLite, Microsoft SQL Server, Oracle, PostgreSQL, dll.



Secara teknis, ketika kita menggunakan **mysqli** extension, PHP langsung berhubungan dengan MySQL Server. Tetapi jika menggunakan PDO, terdapat 1 lapisan (*layer*) di atasnya. PDO hadir sebagai antar muka (*interface*) universal yang menyediakan 1 cara untuk mengakses berbagai jenis database.

Konsep PDO ini dapat digambarkan sebagai berikut:

PHP PDO → Database Driver → Database Server

PDO bekerja dengan metode yang disebut "**data-access abstraction layer**". Artinya, apapun jenis database server yang digunakan, kode PHP yang ditulis tetap sama. PDO lah yang akan menerjemahkan kode tersebut agar bisa dipahami aplikasi database tujuan. Dengan mempelajari cara penggunaan PDO, secara otomatis kita juga bisa membuat kode program untuk berbagai jenis database.

Keunggulan utama PDO ada di satu cara universal dalam **mengakses berbagai jenis database**, bukan untuk berpindah dari satu jenis database ke database lain.

Yang harus dipahami adalah, setiap aplikasi database punya fitur unik yang tidak dimiliki oleh database lain. Jadi meskipun terdapat kode program yang menggunakan PDO, tetap bukan cara yang mudah untuk berpindah dari satu database server ke database server lain (terutama di aplikasi yang sudah jadi).

Di PHP 7, PDO mendukung setidaknya 12 jenis Interface/Database Server:

1. CUBRID
2. Microsoft SQL Server (PDO_SQLSRV)
3. Firebird
4. IBM
5. Informix
6. MySQL
7. MS SQL Server (PDO_DBLIB)
8. Oracle
9. ODBC and DB2
10. PostgreSQL
11. SQLite
12. 4D

Daftar lengkapnya bisa dilihat ke: [PDO Drivers](#).

Mengaktifkan PDO Extension

PDO telah aktif secara default di PHP versi 5.1 ke atas, tetapi tidak semua database driver bisa dipakai. Karena alasan performa, PHP me-nonaktifkan mayoritas PDO database driver seperti Oracle atau PostgreSQL. Untuk melihat driver database apa saja yang telah aktif, jalankan static method

```
PDO::getAvailableDrivers()
```

Coba program berikut, simpan dengan nama pdo1.php, letakkan di folder pertemuan12

```
<?php
print_r(PDO::getAvailableDrivers());
```

Berikut hasil yang saya dapat:

```
Array ( [0] => mysql [1] => sqlite )
```

Terlihat bahwa driver PDO yang aktif hanya ada 2, yakni **MySQL** dan **SQLite**. Artinya, kita hanya bisa memakai PDO untuk mengakses kedua database ini saja. Bagaimana dengan yang lain? harus di aktifkan dari file konfigurasi PHP: php.ini.

Jika anda menginstall XAMPP di drive C, lokasi file php.ini ada di C:\xampp\php\php.ini. Silahkan buka dengan aplikasi text editor, kemudian cari kata "pdo". Pada versi PHP yang saya gunakan, pengaturannya ada di baris 900-an:

```
895 extension=mbstring
896 extension=exif      ; Must be after mbstring as it depends on it
897 extension=mysqli
898 ;extension=oci8_12c  ; Use with Oracle Database 12c Instant Client
899 ;extension=odbc
900 ;extension=openssl
901 ;extension=pdo_firebird
902 extension=pdo_mysql
903 ;extension=pdo_oci
904 ;extension=pdo_odbc
905 ;extension=pdo_pgsql
906 extension=pdo_sqlite
907 ;extension=pgsql
```

Dari gambar di atas, pengaturan *PDO driver extension* ada di baris 902 – 906, yakni baris yang diawali dengan "extension=pdo_", inilah driver database PDO yang tersedia di PHP. Terlihat driver yang telah aktif hanya pdo_mysql dan pdo_sqlite.

Untuk mengaktifkan sebuah driver, **hapus tanda titik koma (;)** di awal baris. Sebagai contoh, saya akan mengaktifkan extension=pdo_pgsql yang merupakan driver untuk database **PostgreSQL**:

```
895 extension=mbstring
896 extension=exif      ; Must be after mbstring as it depends on it
897 extension=mysqli
898 ;extension=oci8_12c  ; Use with Oracle Database 12c Instant Client
899 ;extension=odbc
900 ;extension=openssl
901 ;extension=pdo_firebird
902 extension=pdo_mysql
903 ;extension=pdo_oci
904 ;extension=pdo_odbc
905 extension=pdo_pgsql
906 extension=pdo_sqlite
907 ;extension=pgsql
```

Save file php.ini, kemudian restart web server Apache (matikan dan hidupkan kembali melalui XAMPP Control Panel).

Untuk memastikan apakah driver telah aktif atau belum, jalankan kembali method PDO::getAvailableDrivers() dan berikut adalah hasil yang saya dapat:

```
Array ( [0] => mysql [1] => pgsql [2] => sqlite )
```

Terlihat, driver **PostgreSQL** untuk PDO telah aktif.

Membuat PDO Object

Dalam banyak hal, cara kerja PDO hampir sama dengan mysqli versi object, dimana kita membuat **PDO object** sebagai penghubung dengan database (dikenal sebagai "*database handler*"), lalu menjalankan berbagai method dari object ini.

Berikut format dasar pembuatan PDO object (PDO *constructor*):

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

Terdapat 4 argument yang bisa kita isi pada saat pembuatan PDO object:

- \$dsn: berisi data **DSN**, yakni informasi seputar database server (akan kita bahas sesaat lagi).
- \$username: nama user yang akan mengakses database, misalnya root.
- \$passwd: berisi password dari \$username.
- \$options: berbagai pengaturan tambahan dalam bentuk array.

Selain \$dsn, tiga argumen lain bersifat opsional dan kadang tidak diperlukan untuk driver database tertentu. Misalnya untuk koneksi ke database SQLite tidak perlu menginput \$username dan \$passwd.

Argument pertama pada saat pembuatan PDO object adalah **DSN** (singkatan dari **Data Source Name**).

Berikut contoh penulisan DSN lengkap untuk koneksi ke database MySQL:

```
mysql:host=localhost;port=3306;dbname=ilkoom;charset=utf8mb4
```

DSN di atas bisa dibaca: "Akses driver mysql di localhost dengan nomor port 3306, kemudian langsung pakai database bernama ilkoom dengan character set utf8mb4". Tidak semua pengaturan ini harus ditulis, minimal hanya perlu alamat host saja. Nama database juga tidak harus ditentukan di awal, serta nomor port dan charset bisa memakai nilai default bawaan MySQL.

OK kita langsung coba buat object PDO untuk koneksi ke database, simpan program dengan nama **pdo_connect.php** pada FOLDER **pert_1213**

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", ""); // kita bahas ini
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}
```

Ok kita bahas bagian pembuatan object PDO untuk koneksi

Jadi seperti kalian sudah ketahui semua program diatas membuat sebuah object dari class PDO.

Argument yang dikirim untuk diproses constructor dalam program diatas ada 3

1. DNS : "mysql:host=localhost;dbname=ilkoom"
2. Username database: "root"
3. Password database: ""

Paham ya, apabila program diatas benar saat dijalankan maka proses koneksi akan berjalan sukses.

Nah, untuk bagian exception tidak saya bahas, lagi, intinya sama dengan yang telah kita pelajari di pertemuan sebelumnya.

Bila kalian ingin mencoba melihat bagaimana bila terjadi error koneksi silakan coba dengan memasukkan nilai argument password yang seharusnya kosong dengan nilai tertentu.

OK LALU APA SELANJUTNYA BILA KONEKSI TELAH BERHASIL DIBUAT???

YAK KITA AKAN MULAI MASUK KE PROSES CRUD DATABASE.

Menjalankan Query dengan method PDO::exec()

Proses pembuatan koneksi dengan database sudah selesai, selanjutnya kita akan bahas cara menjalankan perintah query MySQL.

Terdapat beberapa method yang bisa dipakai untuk menjalankan query:

- `PDO::exec()`
- `PDO::query()`
- `PDO::prepare()` dan `PDO::execute()`

Method pertama, yakni `PDO::exec()` hanya bisa dipakai untuk query yang tidak mengembalikan hasil, seperti query `INSERT`, `UPDATE` dan `DELETE`.

Method kedua, yakni `PDO::query()` lebih fleksibel karena bisa dipakai untuk memproses hasil dari query `SELECT`, serta query lain seperti `INSERT`, `UPDATE` dan `DELETE`.

Dan **method ketiga**, yakni `PDO::prepare()` dan `PDO::execute()` dipakai untuk memproses *prepared statement*.

Kita akan bahas ketiga method ini yang dimulai dari `PDO::exec()` terlebih dahulu.

Seperti yang disinggung sebelumnya, method `PDO::exec()` hanya bisa dipakai untuk query yang tidak butuh menampilkan hasil, artinya kita tidak bisa memakai method ini untuk memproses query `SELECT`. Kenapa? Ya karena proses select akan mengembalikan hasil pencarian data.

Sepanjang bab ini saya masih memakai tabel barang yang kita buat pada bab sebelumnya. Apabila kalian telah melakukan beberapa perubahan data silakan coba jalankan program yang awal pernah kita buat untuk input data baru. Atau data boleh disesuaikan, intinya kita akan gunakan tabel barang beserta data yang ada didalamnya.

UPDATE

Ok kali ini kita mulai dari proses update, agar tidak bingung dan repot kalian boleh mengganti data yang digunakan dalam program atau query sesuai dengan data yang ada dalam tabel kalian, misal saya akan mengupdate barang yang id nya 3, bila dalam tabel kalian ga ada barang dengan id 3 silakan ganti dengan id yang lain

Simpan dengan nama `pdo_update1.php`

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $query = "UPDATE barang SET jumlah_barang = 100 WHERE id_barang = 3";
    $count = $pdo->exec($query);
    if ($count !== FALSE) {
        echo "Query Ok, ada $count baris yang di update";
    }
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
```

```
}  
finally {  
    $pdo=NULL;  
}
```

Hasil kode program:

Query Ok, ada 1 baris yang di update

Pembahasan:

Di baris 3 saya membuat PDO object yang langsung mengakses database illoom.

Kemudian di baris 4 terdapat pendefinisian variabel `$query` yang berisi perintah "UPDATE barang SET jumlah_barang = 100 WHERE id_barang = 3". Artinya, saya ingin mengubah data `jumlah_barang` menjadi 100 untuk baris yang memiliki `id_barang = 3`.

Perintah `$pdo->exec($query)` dipakai untuk menjalankan query, yang hasilnya di tampung ke dalam variabel `$count`. Variabel `$count` ini akan berisi FALSE jika terdapat error, atau jumlah *affected rows* apabila ada data yang berhasil di update.

Di baris 6-8 terdapat block `if($count !== FALSE)`. Kondisi ini hanya akan bernilai TRUE jika variabel `$count` berisi nilai selain FALSE, yang berarti query berhasil di proses. Isi dari block `if` sendiri berupa string yang menampilkan jumlah baris terdampak dari hasil query (*affected rows*).

Kita tidak bisa memakai kondisi `if($count)` saja karena ada kemungkinan isi variabel `$count` bernilai 0 yang akan dikonversi PHP menjadi FALSE. Ini terjadi jika query yang dijalankan tidak berdampak apa-apa, misalnya ketika kita menghapus baris yang tidak ada, atau mengupdate baris dengan nilai baru yang sama.

Jika kode di atas dijalankan ulang, hasilnya berupa:

Query Ok, ada 0 baris yang di update

Karena meskipun query UPDATE berhasil di jalankan, tapi tidak ada perubahan nilai di tabel barang.

DELETE

Sebagai contoh kedua, saya ingin menjalankan query DELETE menggunakan method `PDO::exec()`, simpan dengan nama `pdo_delete1.php`:

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $query = "DELET FROM barang WHERE id_barang = 3";
    $count = $pdo->exec($query);
    if ($count !== FALSE) {
        echo "Query Ok, ada $count baris yang di hapus";
    }
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}
```

Hasil kode program:

Query Ok, ada 1 baris yang di hapus

Di sini saya ingin menghapus baris dengan `id_barang = 3` dari tabel barang. Selain perubahan perintah query, tidak ada perbedaan dengan contoh kita sebelumnya.

Error Handling Query

Mari lihat bagaimana PDO menampilkan pesan error untuk query yang salah. Sebagai bahan praktek, tukar isi variabel `$query` dari kode program sebelumnya menjadi:

```
$query = "DELET FROM barang WHERE id_barang = 3";
```

Query ini seharusnya error karena di MySQL tidak terdapat perintah "DELET" (terjadi typo). Bagaimana hasilnya?

Tidak tampil pesan error apapun! Ini merupakan kasus yang sama seperti di **mysqli** object. Tentu saja hal ini bisa membuat pusing karena kita tidak tau apa yang menyebabkan error. Di dalam PDO, ketika sebuah query tidak berhasil dijalankan (error), informasi mengenai pesan error bisa diakses dari method `PDO::errorCode()` dan `PDO::errorInfo()`. Berikut hasil dari pemanggilan kedua method ini:

Ok kita akan melengkapi program dengan menambahkan kondisi jika terjadi error dan menampilkan error apa yang terjadi.

Simpan dengan nama **pdo_error1.php**

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); //tambah ini!!

    $query = "DELET FROM barang WHERE id_barang = 3"; //error terjadi disini "DELET"
    $count = $pdo->exec($query);

    if ($count !== FALSE) {
        echo "Query Ok, ada $count baris yang dihapus";
    }
}
catch (\PDOException $e) { // UBAH BAGIAN INI!!!
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}
```

OK KITA BAHAS BAGIAN

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Dalam materi sebelumnya telah dibahas cara menampilkan pesan error query sebagai *exception*. Kabar baiknya, PDO ternyata memiliki pengaturan khusus yang bisa melempar *exception* secara otomatis. Untuk mengubah pengaturan PDO, terdapat method khusus yang bernama `PDO::setAttribute()`. Method ini butuh 2 argument, yakni aturan yang ingin diubah serta nilai dari aturan tersebut.

Sebagai contoh, berikut perintah agar PDO memproses pesan error query sebagai *exception*:

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Argument pertama, yakni `PDO::ATTR_ERRMODE` adalah keterangan bahwa kita ingin mengubah pengaturan **error mode** di PDO. Terdapat 3 konstanta nilai yang bisa dipilih untuk `PDO::ATTR_ERRMODE`:

- `PDO::ERRMODE_SILENT`
`PDO::ERRMODE_SILENT` adalah pilihan default dimana PDO "menyembunyikan" semua pesan error. Untuk menampilkannya, kita harus akses dari method `PDO::errorCode()` dan `PDO::errorInfo()`.
- `PDO::ERRMODE_WARNING`
`PDO::ERRMODE_WARNING` dipakai untuk menampilkan error sebagai pesan *warning*, kemudian PHP akan lanjut memproses kode program berikutnya.
- `PDO::ERRMODE_EXCEPTION`
`PDO::ERRMODE_EXCEPTION` dipakai untuk menampilkan error sebagai *exception*.

Dari ketiga pilihan ini, kita akan memakai `PDO::ERRMODE_EXCEPTION` agar jika terjadi error di query, PDO otomatis melempar sebuah exception seperti pada program diatas.

Menjalankan Query dengan method `PDO::query()`

Cara kedua untuk menjalankan query di PDO adalah melalui method `PDO::query()`. Berbeda dengan method `PDO::exec()` yang tidak bisa memproses hasil query SELECT, method `PDO::query()` bisa dipakai untuk menjalankan semua perintah query, termasuk SELECT, INSERT, UPDATE, DELETE, dll.

Prinsip kerja dari method `PDO::query()` ini sama seperti method `mysqli::query()`, yakni butuh sebuah argument berupa perintah query MySQL dan mengembalikan sebuah object yang bisa kita proses lebih lanjut.

Jika method `mysqli::query()` mengembalikan **`mysqli_result`** object, maka method `PDO::query()` akan mengembalikan **`PDOStatement`** object. Di dalam **`PDOStatement`** object inilah hasil query seperti data tabel disimpan untuk kemudian bisa di proses dengan berbagai method lanjutan.

Berikut contoh programnya, simpan dengan nama **`pdo_query1.php`**

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "SELECT * FROM barang";
    $stmt = $pdo->query($query); // perhatikan bagian ini !!!
    var_dump($stmt);
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}
```

Hasil kode program:

```
object(PDOStatement)#2 (1) { ["queryString"]=> string(20) "SELECT * FROM barang" }
```

Fokus kita ada di baris 6 – 9. Di baris 6 saya membuat variabel `$query` yang berisi perintah query "SELECT * FROM barang".

Selanjutnya di baris 7 adalah proses pembuatan **PDOStatement** object. Hasil pemanggilan method `$pdo->query($query)` disimpan ke dalam variabel `$stmt`. Variabel `$stmt` inilah yang berisi **PDOStatement** object. Perintah `var_dump($stmt)` di baris 8 untuk menampilkan detail variable `$stmt`.

Ok kita akan coba salah satu fungsi untuk menampilkan jumlah data yang berubah saat menjalankan sebuah query, kita bisa mengakses method `PDOStatement::rowCount()` untuk mengetahui jumlah baris yang terdampak (*affected rows*) saat proses CRUD (INSERT, UPDATE dan DELETE), berikut contohnya simpan dengan nama **pdo_query2.php**:

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "UPDATE barang SET jumlah_barang = 99";
    $stmt = $pdo->query($query);
    if ($stmt !== FALSE) {
        echo "Query Ok, ada ".$stmt->rowCount()." baris yang di update";
    }
    $stmt = NULL;
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}
```

Hasil kode program:

Query Ok, ada 5 baris yang di update

PEMBAHASAN PROGRAM

Di baris 6 saya menjalankan query "UPDATE barang SET jumlah_barang = 99". Query ini akan meng-update kolom jumlah_barang menjadi 99 untuk **seluruh** baris yang ada di dalam tabel barang (karena pada perintah UPDATE ini kita tidak menambahkan kondisi WHERE).

Namun fokus utama kita ada di baris 8 – 10, dimana saya membuat kondisi `if($stmt !== FALSE)`. Kondisi ini akan dijalankan jika tidak ada error di query MySQL, sebab method `query()` di baris 7 akan mengembalikan nilai FALSE jika terdapat error. Jika query tidak error, maka method `$stmt->rowCount()` di baris 8 akan menampilkan jumlah kolom yang terdampak dari hasil query UPDATE tersebut.

Berikutnya, bagaimana menampilkan data yang tersimpan di dalam **PDOStatement** object? Kita bisa memakai salah satu dari method `PDOStatement::fetch`, atau `PDOStatement::fetchAll`.

Silahkan reset ulang tabel barang dengan menjalankan file **create_DB.php** yang sudah kita buat pada pertemuan 8 agar isi tabel barang yang sudah kita ubah berkali-kali kembali seperti semula.

Menampilkan hasil Query dengan `PDOStatement::fetch()`

Method `PDOStatement::fetch()` dipakai untuk memproses hasil dari **PDOStatement** object secara baris per baris. Artinya, jika kita ingin menampilkan seluruh data tabel, method ini harus ditempatkan ke dalam perulangan **while**.

Method `PDOStatement::fetch()` bisa diisi dengan sebuah argument yang akan mengatur seperti apa proses pengambilan data, mirip seperti yang sudah kita pelajari pada materi sebelumnya. Argument ini berbentuk konstanta dengan pilihan sebagai berikut:

- `PDO::FETCH_NUM`: menampilkan data sebagai *numeric array*, dengan index berupa nomor urutan kolom.
- `PDO::FETCH_ASSOC`: menampilkan data sebagai *associative array*, dengan index berupa nama kolom.
- `PDO::FETCH_OBJ`: menampilkan data sebagai object, dengan nama kolom sebagai property.

Selain daftar di atas, masih ada beberapa pilihan lain yang cukup rumit, lengkapnya bisa ke manual PHP di [PDOStatement::fetch](#).

Berikut contoh penggunaan dari `PDOStatement::fetch(PDO::FETCH_NUM)`:
Agar kode program kita lebih ringkas, saya tidak lagi menampilkan seluruh kode program (terutama yang dipakai untuk *error handling*). Simpan dengan nama **pdo_fetch1.php**

```

<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "SELECT * FROM barang";
    $stmt = $pdo->query($query);

    while ($row = $stmt->fetch(PDO::FETCH_NUM)){
        echo $row[0]; echo " | ";
        echo $row[1]; echo " | ";
        echo $row[2]; echo " | ";
        echo $row[3]; echo " | ";
        echo $row[4];
        echo "<br>";
    }
    $stmt = NULL;
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}

```

Hasil kode program:

```

1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-17 15:02:47
2 | Kulkas LG GC-A432HLHU | 10 | 7600000 | 2019-01-17 15:02:47
3 | Laptop ASUS ROG GL503GE | 7 | 16200000 | 2019-01-17 15:02:47
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-17 15:02:47
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-17 15:02:47

```

Terlihat bahwa cara penggunaan method `$stmt->fetch(PDO::FETCH_NUM)` sama seperti method `$result->fetch_row()` di **mysqli** object.

Setiap pemanggilan method `$stmt->fetch(PDO::FETCH_NUM)` akan mengembalikan 1 baris saja, yang dalam contoh ini ditampung ke dalam variabel `$row`. Untuk menampilkan semua baris, harus diproses menggunakan perulangan **while**.

Jika ingin mengambil data tabel sebagai *associative array*, kita bisa memakai method `PDOStatement::fetch(PDO::FETCH_ASSOC)` simpan dengan nama **pdo_fetch2.php**:

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "SELECT * FROM barang";
    $stmt = $pdo->query($query);

    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){
        echo $row['id_barang'];    echo " | ";
        echo $row['nama_barang'];  echo " | ";
        echo $row['jumlah_barang']; echo " | ";
        echo $row['harga_barang']; echo " | ";
        echo $row['tanggal_update'];
        echo "<br>";
    }
    $stmt = NULL;
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}
```

cara di atas mirip seperti method `$result->fetch_assoc()` di **mysqli** object, dimana kita mengakses data tabel dengan nama kolom sebagai key atau index array, jadi jangan bingung ya>>>.

OK, lalu bagaimana jika kita ingin menampilkan data salah satu attribute pada table misal ingin menampilkan data nama barang saja???

Pilihan konstanta lain untuk `PDOStatement::fetch()` adalah `PDO::FETCH_COLUMN`, yang akan mengembalikan array untuk 1 kolom (bukan berbentuk baris). Berikut contoh prakteknya, simpan dengan nama **pdo_fetch3.php**:

```

<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "SELECT nama_barang FROM barang";
    $stmt = $pdo->query($query);

    while ($row = $stmt->fetch(PDO::FETCH_COLUMN)){
        echo $row." <br> ";
    }
    $stmt = NULL;
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}

```

Hasil kode program:

```

TV Samsung 43NU7090 4K
Kulkas LG GC-A432HLHU
Laptop ASUS ROG GL503GE
Printer Epson L220
Smartphone Xiaomi Pocophone F1

```

Di sini query yang saya jalankan adalah "SELECT nama_barang FROM barang", ini dipakai untuk mengambil semua nilai dari kolom nama_barang. Dengan memakai method `fetch(PDO::FETCH_COLUMN)`, variabel `$row` langsung berisi data kolom. Kita tidak perlu lagi menulis judul kolom sebagai *key* array. Jika menggunakan method `$stmt->fetch(PDO::FETCH_ASSOC)`, maka perlu menulis *key* array seperti `$row['nama_barang']`.

Silakan coba untuk menampilkan data yang lain.

Menampilkan hasil Query dengan PDOStatement::fetchAll()

Method PDOStatement::fetchAll() adalah variasi lain dari PDOStatement::fetch() yang baru saja kita pelajari. Melihat dari namanya, bisa di tebak bahwa method fetchAll() dipakai untuk mengambil seluruh data hasil query SELECT, bukan lagi baris per baris sebagaimana fetch().

Array hasil pemanggilan method PDOStatement::fetchAll() berbentuk 2 dimensi karena akan menampung 1 tabel lengkap (memiliki dimensi kolom dan baris). Ini mirip seperti method mysqli_stmt::fetch_all() yang kita bahas pada bab tentang **mysqli** object. Method PDOStatement::fetchAll() juga butuh 1 argument berupa konstanta yang akan menentukan seperti apa array hasil pemanggilan. Berikut beberapa pilihan konstanta tersebut:

- **PDO::FETCH_NUM**
- **PDO::FETCH_ASSOC**
- PDO::FETCH_BOTH
- **PDO::FETCH_OBJ**
- PDO::FETCH_CLASS
- PDO::FETCH_COLUMN
- PDO::FETCH_KEY_PAIR

Kita akan bahas yang saya cetak tebal, Terlihat bahwa semua konstanta yang ada di method PDOStatement::fetch() sebelumnya juga bisa dipakai untuk PDOStatement::fetchAll().

Berikut contoh penggunaan dari PDOStatement::fetchAll(), simpan dengan nama **pdo_fetchAll1.php**:

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "SELECT * FROM barang";
    $stmt = $pdo->query($query);

    $arr = $stmt->fetchAll(PDO::FETCH_NUM); // perhatikan bagian ini
    echo "<pre>";
    print_r($arr);
    echo "</pre>";

    echo "<br>".$arr[1][1];
    $stmt = NULL;
}
```

```

catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}

```

Hasil kode program:

```

Array
(
    [0] => Array
        (
            [0] => 1
            [1] => TV Samsung 43NU7090 4K
            [2] => 5
            [3] => 5399000
            [4] => 2019-01-17 15:02:47
        )
    [1] => Array
        (
            [0] => 2
            [1] => Kulkas LG GC-A432HLHU
            [2] => 10
            [3] => 7600000
            [4] => 2019-01-17 15:02:47
        )
    ...
    ...

Kulkas LG GC-A432HLHU

```

PEMBAHASAN

Di baris 7 saya menjalankan method `$stmt->fetchAll(PDO::FETCH_NUM)` dan menyimpan hasilnya ke dalam variabel `$arr`. Karena menggunakan konstanta `PDO::FETCH_NUM`, maka `$arr` akan berisi **numeric array**.

Perintah `print_r($arr)` di baris 9 memperlihatkan struktur detail dari array `$arr`.

PENTINGG, PAHAMI

Apabila kita ingin mengakses data dari array tersebut maka formatnya adalah `$arr[index][kolom]`. Ingat index array dimulai dari 0 bukan dari 1

Sebagai contoh, bila kita ingin menampilkan isi kolom nama_barang `Kulkas LG GC-A432HLHU` Dimana pada data saya berada pada baris ke 2 (`index[1]`) kolom ke 2 (`kolom[1]`) maka perintahnya adalah `$arr[1][1]`.

bila kita ingin menampilkan isi kolom nama_barang `TV Samsung 43NU7090 4K` Dimana pada data saya berada pada baris ke 1 (`index[0]`) kolom ke 2 (`kolom[1]`) maka perintahnya adalah `$arr[0][1]`.

Jika kita ingin menampilkan semua nilai yang ada di dalam `$arr`, bisa memakai perulangan **foreach**, yang caranya sama seperti di pembahasan tentang **mysqli** object.

Berikut contoh penggunaan dari `PDOStatement::fetchAll(PDO::FETCH_ASSOC)` simpan dengan nama **pdo_fetchAll2.php**:
Sedikit berbeda dalam cara akses data, jangan bingung ya.

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "SELECT * FROM barang";
    $stmt = $pdo->query($query);

    $arr = $stmt->fetchAll(PDO::FETCH_ASSOC);
    echo "<pre>";
    print_r($arr);
    echo "</pre>";

    echo "<br>".$arr[1]["nama_barang"]; //BEDANYA DI SINI !!!!
    $stmt = NULL;
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}
```

Hasil kode program:

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => TV Samsung 43NU7090 4K
            [2] => 5
            [3] => 5399000
            [4] => 2019-01-17 15:02:47
        )
    [1] => Array
        (
            [0] => 2
            [1] => Kulkas LG GC-A432HLHU
            [2] => 10
        )
)
```

```
[3] => 7600000
[4] => 2019-01-17 15:02:47
)
...
...
```

Kulkas LG GC-A432HLHU

Perhatikan perbedaan dengan program sebelumnya hanya pada bagian baris berikut

```
echo "<br>".$arr[1]["nama_barang"];
```

sebelumnya kita akses kolom nama barang dengan menyebut index kolomnya yaitu [1], dengan `fetch_assoc` kita bisa menggunakan nama dari kolom yang ingin kita ambil.

Jika menggunakan konstanta `PDO::FETCH_ASSOC`, index array dimensi pertama tetap berupa nomor (yang menandakan urutan baris), namun untuk dimensi kedua (urutan kolom) akan memakai nama kolom yang berasal dari MySQL.

Mungkin ini akan lebih mudah dan tidak membingungkan, kalian bebas memilih cara mana yang menurut kalian lebih mudah.

Prepared Statement dengan PDO

Agar query yang ditulis lebih aman, terutama jika terdapat data yang berasal dari form, lebih baik menggunakan **prepared statement**. Untungnya, prepared statement di PDO lebih sederhana (dan juga lebih fleksibel) dibandingkan prepared statement versi **mysqli**.

Langkah yang dipakai tetap sama, yakni **prepare** (mempersiapkan query), **bind** (menghubungkan data dengan query) dan **execute** (menjalankan query). Dalam PDO, proses bind dan execute bisa dilakukan dengan 1 perintah saja.

Selain itu PDO tidak butuh object khusus untuk menjalankan prepared statement. Object yang diperlukan tetap **PDOStatement** yang selama ini kita pakai menjalankan method `fetch()` dan `fetchAll()`. Ini berbeda dengan **mysqli** yang butuh **mysqli_stmt** object untuk menjalankan prepared statement.

Karena tidak butuh object baru, maka kita bisa langsung menampilkan hasil prepared statement dengan method `fetch()` dan `fetchAll()` yang telah dibahas sebelumnya.

Berikut contoh penulisan prepared statement dengan PDO simpan dengan nama **preparedPDO1.php**:

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "SELECT * FROM barang WHERE id_barang = ?";    1
    $stmt = $pdo->prepare($query);    2
    $stmt->execute([4]);    3

    $arr = $stmt->fetchAll(PDO::FETCH_NUM); //karena fetch num perhatikan cara aksesnya
    echo "<pre>";
    print_r($arr);
    echo "</pre>";

    echo "<br>".$arr[2][1];
    $stmt = NULL;
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}
```

```

Array
(
    [0] => Array
        (
            [0] => 4
            [1] => Printer Epson L220
            [2] => 14
            [3] => 20990000
            [4] => 2019-01-17 15:02:47
        )
)

```

Printer Epson L220

PENJELASAN PROGRAM, PAHAM! DENGAN SEKSAMA

1. Di baris ini saya membuat query SELECT dengan kondisi WHERE id_barang = ?. Tanda tanya ini nantinya kita input secara terpisah, bila kita sudah memiliki form website maka tanda tanya ini akan diisikan nilai yang di inputkan user melalui form.
2. query yang tersimpan di dalam variabel \$query di jalankan dengan method \$pdo->prepare(\$query), **inilah proses prepare**. Sama seperti pemanggilan method \$pdo->query(), method \$pdo->prepare() ini mengembalikan **PDOStatement object** yang saya simpan ke dalam variabel **\$stmt**.
3. Pemanggilan method **\$stmt->execute([4])** adalah **proses bind** sekaligus **execute** dari prepared statement. Argument yang diisi ke dalam method execute() adalah **nilai pengganti tanda tanya** di query prepared. Dalam hal ini jika saya ingin mengisi angka 4 agar query yang diproses menjadi:

"SELECT * FROM barang WHERE id_barang = 4".

Argument untuk method execute() harus **berbentuk array**, sehingga ditulis sebagai [4]. Apabila data kalian tidak ada barang yang id nya 4, silakan ganti dengan id yang lain.

Bagaimana jika ingin mencari dan menampilkan dengan 2 kondisi? Tidak masalah, cukup tambahkan dua buah tanda tanya dan input 2 buah argument ke dalam method execute() seperti contoh berikut simpan dengan nama **preparedPDO2.php**:

Program dibawah ini kita akan menggunakan method fetch_assoc()

```

<?php
try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "SELECT * FROM barang WHERE id_barang = ? OR
            nama_barang = ?"; // BEDANYA DISINI
    $stmt = $pdo->prepare($query);
    $stmt->execute([1, "Printer Epson L220"]); // BEDANYA DISINI
}

```

```

while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){ // fetch_assoc perhatikan cara aksesnya
    echo $row['id_barang'];    echo " | ";
    echo $row['nama_barang'];  echo " | ";
    echo $row['jumlah_barang']; echo " | ";
    echo $row['harga_barang']; echo " | ";
    echo $row['tanggal_update'];
    echo "<br>";
}

$stmt = NULL;
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}

```

Hasil kode program:

```

1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-17 15:02:47
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-17 15:02:47

```

Kali ini query yang di-prepare adalah

```
"SELECT * FROM barang WHERE id_barang = ? OR nama_barang = ?"
```

Karena ada dua buah tanda tanya, maka array yang diinput ke dalam argument method `execute()` juga perlu 2 buah nilai, yakni `[1, "Printer Epson L220"]`. Dengan demikian, query akhir akan menjadi `SELECT * FROM barang WHERE id_barang = 1 OR nama_barang = "Printer Epson L220"`.

Kemudian saya menggunakan perulangan **while** dan method `$stmt->fetch(PDO::FETCH_ASSOC)` untuk menampilkan hasil tabel barang. Kenapa menggunakan perulangan?? Karena dimungkinkan hasil query lebih dari satu data.

Sampai di sini kita bisa lihat perbedaan antara *prepared statement* di **mysqli** dengan **PDO**. Di PDO, kita tidak butuh proses **bind** secara manual, serta tidak perlu juga menginput jenis tipe data seperti method `bind()` di **mysqli**.

Multiple Execution Prepared Statement

Salah satu keunggulan dari prepared statement (selain keamanan data input), adalah kita mengeksekusi query yang sama lebih dari 1 kali dengan nilai yang berbeda-beda, yakni *multiple execution*.

Materi ini juga sudah kita praktekan di versi **mysqli** object, dan berikut contohnya di dalam PDO, sekalian kita belajar untuk query insert, simpan dengan nama **preparedPDO3.php**:

```
<?php

// Buat format tanggal hari ini
$sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));    1
$timestamp = $sekarang->format("Y-m-d H:i:s");

try {
    $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "INSERT INTO barang (nama_barang, jumlah_barang,
    harga_barang, tanggal_update) VALUES (:nama,:jumlah,:harga,:tanggal);    2
    $stmt = $pdo->prepare($query);

    // Input data 1
    $nama = "Cosmos CRJ-8229 - Rice Cooker";    3
    $jumlah = 4;
    $harga = 299000;
    $tanggal = $timestamp;

    $stmt->execute(['nama'=>$nama, 'jumlah'=>4, 'harga'=>$harga,    4
        'tanggal'=>$tanggal]);
    echo "Query Ok, ".$stmt->rowCount()." baris berhasil ditambah <br>";

    // Input data 2
    $arr_input = [
        'nama' => "Philips Blender HR 2157",
        'jumlah' => 11,
        'harga' => 629000,
```

```

'tanggal' => $timestamp
];

$stmt->execute($arr_input);
echo "Query Ok, ".$stmt->rowCount()." baris berhasil ditambah <br>";
$stmt = NULL;

echo "<hr>";
// Tampilkan data barang
$query = "SELECT * FROM barang";
$stmt = $pdo->query($query);

while ($row = $stmt->fetch(PDO::FETCH_NUM)){
    echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
    echo "<br>";
}
$stmt = NULL;
}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode().")";
}
finally {
    $pdo=NULL;
}

```

5

Hasil kode program:

Query Ok, 1 baris berhasil ditambah

Query Ok, 1 baris berhasil ditambah

1		TV Samsung 43NU7090 4K		5		5399000		2019-01-17 15:02:47
2		Kulkas LG GC-A432HLHU		10		7600000		2019-01-17 15:02:47
3		Laptop ASUS ROG GL503GE		7		16200000		2019-01-17 15:02:47
4		Printer Epson L220		14		2099000		2019-01-17 15:02:47
5		Smartphone Xiaomi Pocophone F1		25		4750000		2019-01-17 15:02:47
6		Cosmos CRJ-8229 - Rice Cooker		4		299000		2019-01-21 07:57:32
7		Philips Blender HR 2157		11		629000		2019-01-21 07:57:32

PEMBAHASAN

OK KITA AKAN BAHAS BEBERAPA BAGIAN PENTING DARI PROGRAM, YANG DALAM LATIHAN-LATIHAN SEBELUMNYA BELUM ADA

1. Di awal program terdapat kode untuk menggenerate tanggal hari ini yang disimpan ke dalam variabel \$timestamp. Nantinya variable ini akan digunakan untuk input data tanggal dalam query
2. Sebelumnya kita memakai tanda tanya " ? " sebagai placeholder atau penanda inputan query untuk prepared statement. Di PDO, terdapat penulisan lain yang dikenal sebagai **named parameter**. Perhatikan perbedaan dengan program sebelumnya.
Dengan named parameter, kita bisa menggunakan "nama" yang diawali dengan tanda titik dua sebagai penanda placeholder, seperti ":id", ":nama", atau ":harga_barang". Kemudian pada saat proses bind, nama ini diisi menggunakan associative array.
3. Perbedaan lain adalah, proses bind di method execute() yang sebelumnya kita lakukan adalah diisi dengan data langsung. Namun jika diinginkan, kita juga bisa mengisinya sebagai variabel seperti contoh diatas.

```
// Input data 1
$nama = "Cosmos CRJ-8229 - Rice Cooker";
$jumlah = 4;
$harga = 299000;
$tanggal = $timestamp;
```

Nah variable yang berisi nilai diatas tinggal di panggil pada proses execute() dipasangkan dengan named parameter yang sudah dibuat diawal.

```
$stmt->execute(['nama'=>$nama, 'jumlah'=>4, 'harga'=>$harga,
'tanggal'=>$tanggal]);
```

4. Merupakan proses execute query, pada program kali ini kita melakukan 2 execute untuk insert 2 buah data sekaligus.
5. Bagian untuk menampilkan semua data dari table barang yang kita miliki.

Latihan terakhir

Membuat Table mahasiswaSV dengan konsep PDO pada database ilkoom
kode yang perlu dibuat adalah:

1. Buat koneksi dengan MySQL menggunakan PDO.
2. Buat database ilkoom jika belum ada (CREATE DATABASE).
3. Hapus tabel mahasiswa jika ada (DROP DATABASE).
4. Buat tabel mahasiswa (CREATE TABLE).
5. Isi tabel mahasiswa (INSERT).
6. Tampilkan isi tabel mahasiswa (SELECT).

Tampilan data mahasiswaSV

NIM	Nama	Tempat Lahir	Tanggal Lahir	Fakultas	Jurusan	IPK
13012012	James Situmorang	Medan	02 - 04 - 1995	Kedokteran	Kedokteran Gigi	2.70
14005011	Riana Putria	Padang	23 - 11 - 1996	FMIPA	Kimia	3.10
15002032	Rina Kumala Sari	Jakarta	28 - 06 - 1997	Ekonomi	Akuntansi	3.40
15021044	Rudi Permana	Bandung	22 - 08 - 1994	FASILKOM	Ilmu Komputer	2.90
15003036	Sari Citra Lestari	Jakarta	31 - 12 - 1997	Ekonomi	Manajemen	3.50

Berikut programnya simpan dengan nama **lastPDO.php**

```
<?php
try {
    $pdo = new PDO("mysql:host=localhost", "root", "");
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Buat database "ilkoom" (jika belum ada)
    $query = "CREATE DATABASE IF NOT EXISTS ilkoom";
    $stmt = $pdo->query($query);
    if ($stmt !== FALSE){
        echo "Database 'ilkoom' berhasil di buat / sudah tersedia <br>";
    }

    // Pilih database "ilkoom"
    $query = "USE ilkoom";
    $stmt = $pdo->query($query);
    if ($stmt !== FALSE){
        echo "Database 'ilkoom' berhasil di pilih <br>";
    }

    // Hapus tabel "mahasiswaSV" (jika ada)
```

```

$query = "DROP TABLE IF EXISTS mahasiswaNF";
$pdo->query($query);

// Buat tabel "mahasiswaSV"
$query = "CREATE TABLE mahasiswaSV (
    nim CHAR(8) PRIMARY KEY,
    nama VARCHAR(100),
    tempat_lahir VARCHAR(50),
    tanggal_lahir DATE,
    fakultas VARCHAR(50),
    jurusan VARCHAR(50),
    ipk DECIMAL(3,2))";
$stmt = $pdo->query($query);
if ($stmt !== FALSE){
    echo "Tabel 'mahasiswaSV' berhasil di buat <br>";
}

// Isi tabel "mahasiswaSV"
$query = "INSERT INTO mahasiswaSV VALUES
    ('14005011', 'Riana Putria', 'Padang', '1996-11-23', 'FMIPA', 'Kimia', 3.1),
    ('15021044', 'Rudi Permana', 'Bandung', '1994-08-
22', 'FASILKOM', 'Ilmu Komputer', 2.9),
    ('15003036', 'Sari Citra Lestari', 'Jakarta', '1997-12-31', 'Ekonomi', 'Manajemen', 3.5),
    ('15002032', 'Rina Kumala Sari', 'Jakarta', '1997-06-28', 'Ekonomi', 'Akuntansi', 3.4),
    ('13012012', 'James Situmorang', 'Medan', '1995-04-
02', 'Kedokteran', 'Kedokteran Gigi', 2.7)";
$stmt = $pdo->query($query);
if ($stmt !== FALSE){
    echo "Tabel 'mahasiswaSV' berhasil di isi ".$stmt->rowCount().
    " baris data <br>";
}

echo "<hr>";

```

```
// Tampilkan semua isi tabel "mahasiswaSV"
echo "<h3>Tabel Mahasiswa</h3>";
$query = "SELECT * FROM mahasiswaSV";
$stmt = $pdo->query($query);

while ($row = $stmt->fetch(PDO::FETCH_NUM)){
    echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
    echo "<br>";
}
$stmt = NULL;

}
catch (\PDOException $e) {
    echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (". $e->getCode(). ")";
}
finally {
    $pdo=NULL;
}
}
```

Hasil

Most Visited Check IMEI - IMEI.info Getting Started SuperSoccer.co.id di					
Database 'ilkoom' berhasil di buat / sudah tersedia Database 'ilkoom' berhasil di pilih Tabel 'mahasiswaSV' berhasil di buat Tabel 'mahasiswaSV' berhasil di isi 5 baris data					
Tabel Mahasiswa					
13012012	James Situmorang	Medan	1995-04-02	Kedokteran	
14005011	Riana Putra	Padang	1996-11-23	FMIPA	
15002032	Rina Kumala Sari	Jakarta	1997-06-28	Ekonomi	
15003036	Sari Citra Lestari	Jakarta	1997-12-31	Ekonomi	
15021044	Rudi Permana	Bandung	1994-08-22	FASILKOM	

Selesai, sampai bertemu di matakuliah selanjutnya 😊