

10. 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. Dalam bab ini kita akan kupas secara mendalam apa itu PDO serta bagaimana cara penggunaannya.

10.1. 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.



Ilustrasi PDO (sumber gambar: cloudways.com)

Secara teknis, ketika kita menggunakan **mysqli** extension, PHP langsung terhubung 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, tetapi bukan cara yang mudah untuk berpindah dari satu database server ke database server lain (terutama di aplikasi yang sudah jadi).

Di PHP 8, PDO mendukung setidaknya 11 jenis Interface/Database Server:

1. CUBRID
2. MS SQL Server
3. Firebird
4. IBM
5. Informix
6. MySQL
7. MS SQL Server
8. Oracle
9. ODBC and DB2
10. PostgreSQL
11. SQLite

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

10.2. 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()`:

```
01.pdo_getavailabledrivers.php
1 <?php
2     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 diaktifkan 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

```

Pengaturan PDO driver extension

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

```

Aktifkan `pdo_pgsql` driver extension

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.

Meskipun driver PDO untuk sebuah database telah aktif, tetap tidak bisa langsung

dipakai karena kita juga harus menginstall aplikasi database tersebut.

Sebagai contoh, agar bisa mengakses database PostgreSQL dari PHP, kita harus menginstall aplikasi PostgreSQL server terlebih dahulu dan menjalankannya.

Karena keterbatasan tempat untuk membahas cara instalasi PostgreSQL (serta aplikasi database lain), dalam materi ini kita hanya membahas cara pemakaian PDO untuk database MySQL atau MariaDB saja.

10.3. 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**). DSN ini berbentuk string dengan format:

```
nama_driver_pdo:<pengaturan_khusus_driver>
```

Nama driver PDO adalah nama driver yang di dapat dari hasil pemanggilan method `PDO::getAvailableDrivers()`. Misalnya untuk database MySQL, nama drivernya adalah "`mysql`", sedangkan untuk PostgreSQL, nama drivernya adalah "`pgsql`".

Setelah nama driver, disambung dengan tanda titik dua ":" , kemudian diikuti dengan pengaturan lain tergantung driver yang dipakai.

Untuk MySQL, informasi yang harus dicantumkan adalah alamat komputer (**host**) tempat MySQL Server berada, yang dalam contoh kita berupa `localhost`. Pengaturan opsional lain

berupa nama port (`port`), nama database (`dbname`), serta character set (`charset`) yang akan dipakai.

Pengaturan ini ditulis dalam format `nama_pengaturan=nilai`, yang dipisah dengan tanda titik koma. Berikut contoh penulisan DSN lengkap untuk koneksi ke database MySQL:

```
mysql:host=localhost;port=3306;dbname=ilkoom;charset=utf8mb
```

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.

Dengan demikian, berikut contoh pembuatan **PDO** object untuk mengakses database MySQL:

02.pdo_connect.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost", "root", "");
3 var_dump($pdo); // object(PDO)#1 (0) { }
```

Variabel `$pdo` saya pakai sebagai penampung dari **PDO** object. Nama variabel ini boleh bebas, tidak harus `$pdo`. PDO object ini berisi koneksi ke database MySQL yang ada di `localhost`, kemudian masuk sebagai user `root` dengan password kosong.

Perintah `var_dump($pdo)` di baris 3 saya tambah untuk melihat bahwa variabel `$pdo` berisi sebuah `object(PDO)`.

Jika kita ingin menulis DSN lengkap juga tidak masalah:

03.pdo_connect_full_DSN.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;port=3306;dbname=ilkoom;charset=utf8mb4",
3                 "root", "");
4 var_dump($pdo); // object(PDO)#1 (0) { }
```

Kode di atas artinya saya ingin masuk ke MySQL server yang berada di `localhost` dengan nomor port `3306`, langsung memakai database `ilkoom`, menggunakan charset `utf8mb4`, serta login sebagai user `root` dengan password kosong.

Karena cukup panjang, penulisan argument untuk PDO bisa dipisah menjadi variabel tersendiri agar lebih rapi:

04.pdo_connect_variable.php

```
1 <?php
2 $host    = "127.0.0.1";
3 $port    = "3306";
```

```

4 $db      = "ilkoom";
5 $charset = "utf8mb4";
6 $user    = "root";
7 $pass    = "";
8
9 $dsn = "mysql:host=$host;port:$port;dbname=$db;charset=$charset";
10 $pdo = new PDO($dsn, $user, $pass);

```

Dari segi isi, tidak ada perbedaan dengan contoh kita sebelumnya. Hanya saja kali ini pembuatan PDO object tampak lebih rapi karena dipecah menjadi variabel-variabel.

Namun demi menghemat tempat, dalam contoh selanjutnya saya "terpaksa" memakai versi yang 1 baris saja.

Walaupun kita hanya membahas cara penggunaan PDO ke database MySQL, berikut contoh pembuatan koneksi untuk database **Microsotf SQL Server**, **Sybase** dan **SQLite**:

```

$pdo = new PDO("mssql:host=$host;dbname=$dbname, $user, $pass");
$pdo = new PDO("sybase:host=$host;dbname=$dbname, $user, $pass");
$pdo = new PDO("sqlite:my/database/path/database.db");

```

Terlihat bahwa argument pembuatan PDO object bisa berbeda antar aplikasi database. Untuk SQLite tidak perlu menulis user dan password, tapi cukup alamat ke lokasi file database SQLite saja.

Setelah PDO object selesai digunakan, kita bisa menutupnya dengan mengisi nilai **NULL** ke dalam variabel penampung PDO, seperti contoh berikut:

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost", "root", "");
3 //...
4 //...
5 $pdo = NULL;

```

Yup, PDO tidak menyediakan method khusus untuk menutup koneksi seperti halnya **mysqli::close()**, jadi terpaksa dibuat manual dengan cara mengisi nilai **NULL**.

Proses pemberian nilai **NULL** ini sebenarnya juga tidak harus ditulis karena PHP otomatis menutup koneksi begitu halaman selesai di proses.

10.4. Error handling PDO Object

Penanganan error pada saat pembuatan PDO object butuh perhatian khusus. Karena secara default jika koneksi ke database tidak bisa di proses, PHP akan menampilkan pesan error yang di dalamnya terdapat seluruh informasi, termasuk nama user dan password!

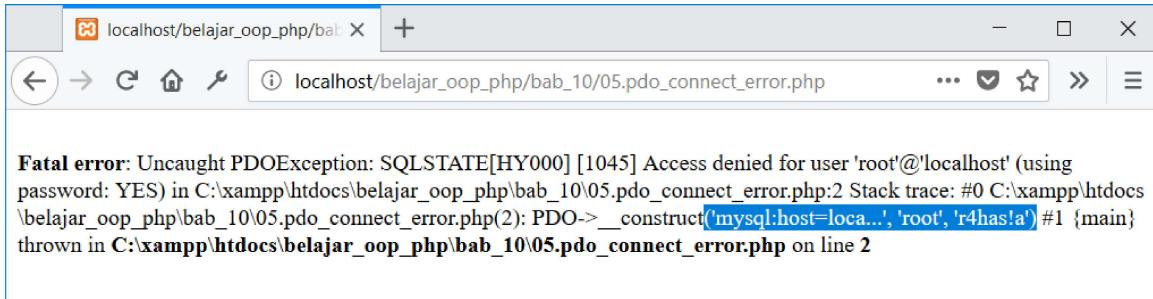
Berikut contoh kasusnya:

05 pdo_connect_error.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "r4has!a");

```



Tampilan pesan error pada saat pembuatan PDO Object

Pada saat pembuatan PDO object, saya mengisi password user root dengan string "r4has!a". Hasilnya tampil error karena user root seharusnya tanpa password. Namun yang jadi masalah adalah, dalam pesan error ini terlihat informasi sensitif berupa semua argument pembuatan PDO object. Tentu saja ini sangat berbahaya.

Solusi yang paling pas adalah mematikan *error reporting* agar semua pesan error tidak lagi bisa terlihat. Ini bisa dilakukan dari file `php.ini` atau dengan menjalankan fungsi `error_reporting(0)` di baris paling atas file PHP. Namun pilihan ini baru pas dilakukan saat aplikasi yang kita buat sudah selesai.

Alternatif lain adalah menggunakan block **try – catch**. Jika anda perhatikan, pesan error di atas diawali dengan `Fatal error: Uncaught PDOException`. Ini adalah pesan error ketika sebuah exception tidak ditangkap. Artinya, secara bawaan PDO sudah "melempar" sebuah exception jika terjadi error. Kita tinggal menangkap `PDOException` ini:

06 pdo_connect_try_catch.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "r4has!a");
4 }
5 catch (\PDOException $e) {
6     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
7 }
8 finally {
9     $pdo=NULL;
10}

```



Tampilan pesan error PDO dengan exception handling

Dengan menggunakan block try-catch, kita bisa mengontrol hasil tampilan error. Dalam contoh ini saya hanya menampilkan informasi dari `$e->getMessage()` yang berisi pesan error dan `$e->getCode()` yang berisi nomor kode error. Ini sama seperti yang kita pakai dalam bab **mysqli** object sebelumnya.

Class exception yang dihasilkan pada saat pembuatan PDO bernama `PDOException`, awalan tanda "\\" pada block catch (`\PDOException $e`) adalah kode untuk *global namespace*. Dalam contoh ini sebenarnya tidak perlu ditulis karena kita sedang berada di global namespace (saya tidak menggunakan namespace apapun). Jika berada di dalam namespace, awalan "\\" ini perlu ditulis.

10.5. 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`.

Method `PDO::exec()` butuh sebuah argument berupa perintah query yang akan dijalankan. Kemudian method ini mengembalikan salah satu dari 2 nilai:

- ◆ **False**, jika perintah query yang ditulis terdapat error.
- ◆ **Angka integer**, berisi jumlah baris yang dipengaruhi oleh perintah query (*affected rows*).

Berikut contoh penggunaannya:

Sepanjang bab ini saya masih memakai tabel `barang` yang kita buat pada bab sebelumnya.
Silahkan reset ulang dengan menjalankan file `bab09\20.mysql_generate.php`.

07 pdo_exec_update.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $query = "UPDATE barang SET jumlah_barang = 100 WHERE id_barang = 3";
5     $count = $pdo->exec($query);
6     if ($count !== FALSE) {
7         echo "Query Ok, ada $count baris yang di update";
8     }
9 }
10 catch (\PDOException $e) {
11     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
12 }
13 finally {
14     $pdo=NULL;
15 }
```

Hasil kode program:

Query Ok, ada 1 baris yang di update

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

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 tumpung ke dalam variabel `$count`. Variabel `$count` ini akan berisi `FALSE` jika terdapat error, atau jumlah *affected rows*.

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`.

Sebagai contoh kedua, saya ingin menjalankan query `DELETE` menggunakan method `PDO::exec()`:

08.pdo_exec_delete.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $query = "DELETE FROM barang WHERE id_barang = 3";
5     $count = $pdo->exec($query);
6     if ($count !== FALSE) {
7         echo "Query Ok, ada $count baris yang di hapus";
8     }
9 }
10 catch (\PDOException $e) {
11     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()."";
12 }
13 finally {
14     $pdo=NULL;
15 }
```

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.

10.6. Error Handling Query

Salah satu perubahan di PHP 8 berhubungan dengan tampilan pesan error query di PDO.

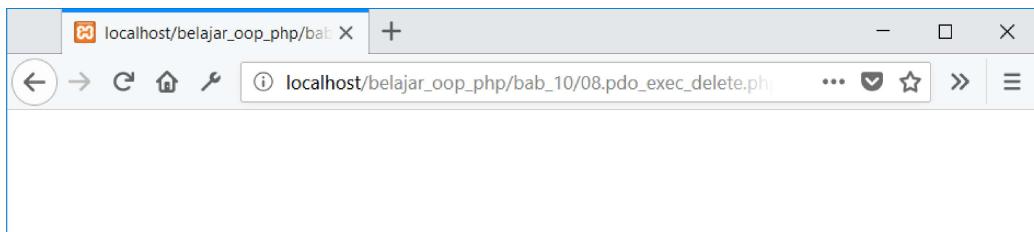
Sebelumnya di PHP 7, error query PDO tidak langsung tampil (harus kita aktifkan secara manual). Sedangkan di PHP 8 secara default pesan error PDO sudah langsung tampil. Perbedaan ini sebenarnya tidak terlalu berdampak karena tetap bisa diatur sesuai keinginan.

Oleh karena itu bagi yang sudah menggunakan PHP 8, bahasan ini boleh dilewati. Bagian ini khusus bagi yang memakai PHP 7 atau PHP 5.6 saja.

Di PHP 7, secara default PDO menyembunyikan pesan error query. Untuk membuktikan, silahkan 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". Bagaimana hasilnya?



Tampilan default PDO jika terjadi query error

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.

Jika menggunakan PHP 8, kode di atas akan menampilkan 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:

09 pdo_exec_error_info.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $query = "DELET FROM barang WHERE id_barang = 3";
5     $count = $pdo->exec($query);
6
7     echo "<pre>";
8     var_dump($pdo->errorCode());
9     var_dump($pdo->errorInfo());
10    echo "</pre>";
11
12    if ($count !== FALSE) {
13        echo "Query Ok, ada $count baris yang dihapus";
14    }
15 }
16 catch (\PDOException $e) {
17     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
18 }
19 finally {
20     $pdo=NULL;
21 }
```

Hasil kode program:

```

string(5) "42000"
array(3) {
[0]=>
string(5) "42000"
```

```
[1]=>
int(1064)
[2]=>
string(185) "You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to use near 'DELET
FROM barang WHERE id_barang = 3' at line 1"
}
```

Kembali, jika kode ini dijalankan pada PHP 8, hasilnya sedikit berbeda. Tampilan di atas hanya terlihat di PHP 7 dan PHP 5.6.

Di baris 8 dan 9 saya menggunakan perintah `var_dump()` untuk melihat hasil dari `$pdo->errorCode()` dan `$pdo->errorInfo()`.

Method `$pdo->errorCode()` menghasilkan kode error dalam format `SQLSTATE`, yakni 5 digit kode error alfanumerik. Kode ini bersifat universal untuk semua aplikasi database. Jika anda tertarik, arti dari setiap kode `SQLSTATE` bisa diakses ke en.wikipedia.org/wiki/SQLSTATE.

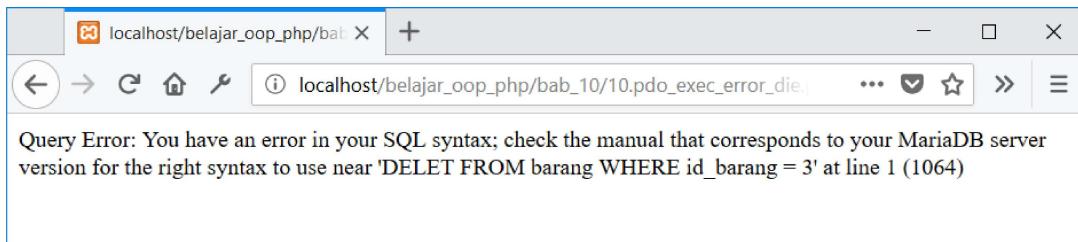
Method `$pdo->errorInfo()` mengembalikan nilai dalam bentuk array 3 element:

1. Element pertama berisi kode `SQLSTATE`, yakni sama dengan hasil pemanggilan `$pdo->errorCode()`.
2. Element kedua berisi nomor error milik MySQL.
3. Element ketiga berisi keterangan error dari MySQL.

Dari ketiga element ini, informasi error yang perlu kita ketahui adalah kode error MySQL dan pesan error MySQL, yakni element ke-2 dan ke-3. Berikut revisi kode program sebelumnya:

10.pdo_exec_error_die.php

```
1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $query = "DELETE FROM barang WHERE id_barang = 3";
5     $count = $pdo->exec($query);
6
7     if ($count !== FALSE) {
8         echo "Query Ok, ada $count baris yang dihapus";
9     }
10    else {
11        die("Query Error: ".$pdo->errorInfo()[2]." (".$pdo->errorInfo()[1].")");
12    }
13 }
14 catch (\PDOException $e) {
15     echo "Koneksi / Query bermasalah: ".$e->getMessage()." (".$e->getCode().")";
16 }
17 finally {
18     $pdo=NULL;
19 }
```



Hasil error dari PDO

Di baris 10 – 12 saya menambah block **else** dari kondisi `if($count !== FALSE)`. Artinya, blok **else** hanya dijalankan jika terdapat error di penulisan query. Isinya berupa fungsi `die()` untuk menghentikan kode program dan menampilkan pesan error yang tersimpan di `$pdo->errorInfo()[2]` serta kode error di `$pdo->errorInfo()[1]`.

Jika anda mengikuti materi di bab sebelumnya tentang **mysqli**, saya yakin bisa menebak arah dari kode pembahasan kita. Yup... daripada menggunakan fungsi `die()`, lebih baik pesan error ini diproses sebagai sebuah **exception**. Lagi pula kita sudah memiliki block kode **catch** di akhir kode program:

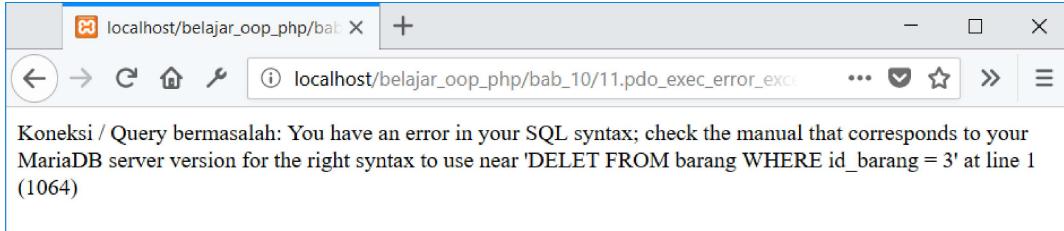
`11 pdo_exec_error_exception.php`

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $query = "DELET FROM barang WHERE id_barang = 3";
5     $count = $pdo->exec($query);
6
7     if ($count !== FALSE) {
8         echo "Query Ok, ada $count baris yang dihapus";
9     }
10    else {
11        throw new Exception($pdo->errorInfo()[2], $pdo->errorInfo()[1]);
12    }
13 }
14 catch (\Exception $e) {
15     echo "Koneksi / Query bermasalah: ". $e->getMessage(). " (" . $e->getCode() . ")";
16 }
17 finally {
18     $pdo=NULL;
19 }
```

Sekarang di dalam block **else** terdapat perintah untuk membuat exception. Sebagai argument, diinput pesan error yang tersimpan di `$pdo->errorInfo()[2]` dan `$pdo->errorInfo()[1]`.

Di baris 14 saya menukar kondisi **catch** dari sebelumnya `catch(\PDOException $e)` menjadi `catch(\Exception $e)`. Ini bertujuan agar block **catch** bersifat global dan bisa menangkap semua exception.



Hasil error PDO dengan exception yang dibuat manual

Harap dibedakan bahwa exception yang kita buat di sini dipakai untuk menampilkan **pesan kesalahan dari perintah query**. Sedangkan exception yang kita bahas di awal bab adalah pesan kesalahan pada saat **pembuatan PDO object**.

10.7. Pengaturan PDO dengan method PDO::setAttribute()

Cara menampilkan pesan error query yang kita bahas sebelumnya secara khusus ditujukan untuk PHP 7 ke bawah. Sedangkan di PHP 8 tidak perlu lagi karena exception sudah "dilempar" secara otomatis.

Perbedaan ini berkaitan dengan pengaturan PDO yang bisa di-set dari method khusus bernama `PDO::setAttribute()`. Method `PDO::setAttribute()` 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` berisi 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_WARNING`
- ◆ `PDO::ERRMODE_EXCEPTION`

`PDO::ERRMODE_SILENT` adalah pilihan default di PHP 7, dimana PDO "menyembunyikan" semua pesan error. Untuk menampilkannya, kita harus akses dari method `PDO::errorCode()` dan `PDO::errorInfo()` seperti contoh sebelumnya.

`PDO::ERRMODE_WARNING` dipakai untuk menampilkan error sebagai pesan *warning*, kemudian PHP akan lanjut memproses kode program berikutnya.

`PDO::ERRMODE_EXCEPTION` dipakai untuk menampilkan error sebagai exception. Inilah yang menjadi pengaturan default di PHP 8 sehingga error query sudah langsung tampil.

Dari ketiga pilihan ini, sebaiknya set ke `PDO::ERRMODE_EXCEPTION` agar jika terjadi error query,

PDO otomatis melempar sebuah exception. Berikut contoh prakteknya:

12.pdo_exec_error_set_exception.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5
6     $query = "DELETE FROM barang WHERE id_barang = 3";
7     $count = $pdo->exec($query);
8
9     if ($count !== FALSE) {
10        echo "Query Ok, ada $count baris yang dihapus";
11    }
12 }
13 catch (\PDOException $e) {
14     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
15 }
16 finally {
17     $pdo=NULL;
18 }
```

Di baris 4 terdapat pemanggilan method `$pdo->setAttribute()` yang berisi pengaturan agar PDO memproses pesan error query sebagai exception.

Dengan demikian, kita tidak butuh lagi kode untuk membuat exception secara manual di PHP 7. Jika query yang dijalankan method `$pdo->exec()` salah ketik atau tidak dipahami oleh MySQL, PDO langsung melempar sebuah exception.

Exception yang dilempar berasal dari class `\PDOException`, yakni sama seperti exception yang dipakai saat pembuatan PDO object. Sehingga kita bisa kembali menulis block **catch** sebagai `catch(\PDOException $e)` di baris 13.

Sebagai alternatif penulisan, pengaturan PDO ini juga bisa ditulis sebagai argument ke-4 pada saat pembuatan PDO object (PDO constructor). Berikut cara penulisannya:

13.pdo_exec_error_set_exception_constuctor.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "",
4                     [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]);
5 // ...
6 // ...
```

Argument ke-4 ini ditulis dalam bentuk *associative array*, yakni dengan format `<nama_pengaturan> => <nilai_pengaturan>`. Jika terdapat beberapa pengaturan, pisah dengan tanda koma sebagaimana sebuah array:

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "",
4                     [ PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
5                       PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC ]);
6 // ...
7 // ...

```

Maksud dari `PDO::ATTR_DEFAULT_FETCH_MODE` akan kita bahas sesaat lagi. Untuk sementara yang perlu dipahami adalah bahwa pengaturan PDO bisa ditulis sebagai argument ke-4 pada saat pembuatan PDO object, atau memakai method `PDO::setAttribute()`.

10.8. 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 kita proses dengan berbagai method lanjutan.

Apabila query yang ditulis error atau tidak bisa dipahami oleh MySQL, method `PDO::query()` akan mengembalikan nilai boolean `FALSE` dan otomatis melempar exception jika pengaturan `PDO::ERRMODE_EXCEPTION` aktif.

Berikut contoh pembuatan `PDOStatement` object:

14.pdo_query_stmt_obj.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
5
6     $query = "SELECT * FROM barang";
7     $stmt = $pdo->query($query);
8     var_dump($stmt);
9     $stmt = NULL;
10 }
11 catch (\PDOException $e) {
12     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()."";

```

```

13 }
14 finally {
15     $pdo=NULL;
16 }
```

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 memperlihatkan hal ini.

Di baris 9 saya mengisi nilai NULL ke dalam variabel \$stmt. Ini adalah cara untuk menghapus **PDOStatement** object jika sudah tidak diperlukan lagi. Sebenarnya ini sama seperti method \$mysqli_result::free(), namun karena PDO tidak memiliki method seperti itu maka terpaksa di hapus manual dengan cara mengisi nilai NULL.

Kembali, proses penghapusan **PDOStatement** object ini bersifat opsional dan tidak harus ditulis. Artinya, variabel penampung **PDOStatement** object (\$stmt) tidak harus diisi nilai NULL di akhir kode program.

Apabila method **PDO**::**query()** dipakai untuk menjalankan perintah MySQL yang mengubah data (**INSERT**, **UPDATE** dan **DELETE**), kita bisa mengakses method **PDOStatement**::**rowCount()** untuk mengetahui jumlah baris yang terdampak (*affected rows*), berikut contohnya:

14a.pdo_query_stmt_obj_rowcount.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5
6     $query = "UPDATE barang SET jumlah_barang = 99";
7     $stmt = $pdo->query($query);
8     if ($stmt !== FALSE) {
9         echo "Query Ok, ada ".$stmt->rowCount()." baris yang di update";
10    }
11    $stmt = NULL;
12 }
13 catch (\PDOException $e) {
14     echo "Koneksi / Query bermasalah: ".$e->getMessage()." (".$e->getCode().")";
15 }
16 finally {
17     $pdo=NULL;
18 }
```

Hasil kode program:

```
Query Ok, ada 5 baris yang di update
```

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 perintah UPDATE ini tidak memiliki 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 `bab09\20.mysql_generate.php` agar isi kolom `jumlah_barang` kembali seperti semula.

10.9. 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 satu argument yang akan mengatur seperti apa proses pengambilan data. Argument ini berbentuk konstanta dengan berbagai pilihan:

- ◆ `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_BOTH`: menampilkan data sebagai *numeric array* dan *associative array* sekaligus. Ini adalah pengaturan default jika method `fetch()` dipanggil tanpa argument.
- ◆ `PDO::FETCH_OBJ`: menampilkan data sebagai object, dengan nama kolom sebagai property.
- ◆ `PDO::FETCH_LAZY`: menampilkan data sebagai *numeric array*, *associative array*, dan object sekaligus.
- ◆ `PDO::FETCH_COLUMN`: menampilkan 1 kolom data.

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*). Versi lengkapnya bisa anda buka dari file [belajar_oop_php.zip](#)

15.pdo_query_fetch_num.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
8     echo $row[0];    echo " | ";
9     echo $row[1];    echo " | ";
10    echo $row[2];   echo " | ";
11    echo $row[3];   echo " | ";
12    echo $row[4];
13    echo "<br>";
14 }
```

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)`:

16.pdo_query_fetch_assoc.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
```

PDO

```
6
7 while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){
8     echo $row['id_barang'];      echo " | ";
9     echo $row['nama_barang'];    echo " | ";
10    echo $row['jumlah_barang'];  echo " | ";
11    echo $row['harga_barang'];   echo " | ";
12    echo $row['tanggal_update'];
13    echo "<br>";
14 }
```

Cara ini mirip seperti method `$result->fetch_assoc()` di **mysqli** object, dimana kita mengakses data tabel dengan nama kolom sebagai key atau index array.

Selanjutnya, berikut contoh penggunaan dari `PDOStatement::fetch(PDO::FETCH_BOTH)`:

17.pdo_query_fetch_both.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 while ($row = $stmt->fetch(PDO::FETCH_BOTH)){
8     echo $row['id_barang'];      echo " | ";
9     echo $row[1];               echo " | ";
10    echo $row['jumlah_barang'];  echo " | ";
11    echo $row[3];               echo " | ";
12    echo $row['tanggal_update'];
13    echo "<br>";
14 }
```

Dengan memakai konstanta `PDO::FETCH_BOTH`, kita bisa mengakses array `$row` menggunakan *numeric array* maupun *associative array*. Ini adalah pilihan default jika method `fetch()` dipanggil tanpa menulis argument seperti contoh berikut:

```
1 ...
2 while($row = $stmt->fetch()){
3 ...
```

Nantinya kita juga bisa mengubah pengaturan default ini.

Jika menggunakan konstanta `PDO::FETCH_OBJ`, maka proses menampilkan data tabel akan memakai pemanggilan object:

18.pdo_query_fetch_obj.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
```

PDO

```
7 while ($row = $stmt->fetch(PDO::FETCH_OBJ)){
8     echo $row->id_barang;      echo " | ";
9     echo $row->nama_barang;    echo " | ";
10    echo $row->jumlah_barang;  echo " | ";
11    echo $row->harga_barang;   echo " | ";
12    echo $row->tanggal_update;
13    echo "<br>";
14 }
```

Cara menampilkan ini sama seperti method `$result->fetch_object()` di versi **mysqli**, dimana kita memakai format `$row->nama_kolom`.

Yang tidak ada di versi **mysqli** adalah, mengakses *numeric array*, *associative array* dan *object* sekaligus. Di PDO ini bisa dilakukan dengan method `PDOStatement::fetch(PDO::FETCH_LAZY)`:

19.pdo_query_fetch_lazy.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 while ($row = $stmt->fetch(PDO::FETCH_LAZY)){
8     echo $row['id_barang'];      echo " | ";
9     echo $row[1];                echo " | ";
10    echo $row->jumlah_barang;  echo " | ";
11    echo $row[3];                echo " | ";
12    echo $row->tanggal_update;
13    echo "<br>";
14 }
```

Di sini saya mengakses variabel `$row` dengan 3 cara: *numeric array*, *associative array* serta *object*.

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

20.pdo_query_fetch_column.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT nama_barang FROM barang";
5 $stmt = $pdo->query($query);
6
7 while ($row = $stmt->fetch(PDO::FETCH_COLUMN)){
8     echo $row." | ";
9 }
```

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']`.

Sebelumnya telah dijelaskan bahwa konstanta `PDO::FETCH_BOTH` adalah pilihan default jika method `fetch()` dipanggil tanpa argument. Kita bisa mengubah pengaturan ini dari method `PDO::setAttribute()`. Caranya, gunakan `PDO::ATTR_DEFAULT_FETCH_MODE` sebagai argument pertama, lalu pilih salah satu konstanta method `fetch()` sebagai argument kedua.

Sebagai contoh, jika saya ingin pilihan default method `fetch()` adalah `PDO::FETCH_LAZY`, maka kode programnya adalah sebagai berikut:

21.pdo_query_fetch_setattribute.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3 $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
4 $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_LAZY);
5
6 $query = "SELECT * FROM barang";
7 $stmt = $pdo->query($query);
8
9 while ($row = $stmt->fetch()){
10    echo $row['id_barang'];
11    echo $row[1];
12    echo $row->jumlah_barang;
13    echo $row[3];
14    echo $row->tanggal_update;
15    echo "<br>";
16 }
```

Dengan tambahan kode program di baris 4, maka ketika method `fetch()` dipanggil tanpa argument seperti di baris 9, secara otomatis menggunakan konstanta `PDO::FETCH_LAZY`.

Alternatif penulisan lain adalah pada saat pembuatan PDO object, seperti:

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "",
4                     [ PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
5                       PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_OBJ ]);
6 // ...
7 // ...
```

Sekarang jika method `PDOStatement::fetch()` dipanggil tanpa argument, akan memakai konstanta `PDO::FETCH_OBJ` secara default.

10.10. 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`

Terlihat bahwa semua konstanta yang ada di method `PDOStatement::fetch()` sebelumnya juga bisa dipakai untuk `PDOStatement::fetchAll()`, kecuali `PDO::FETCH_LAZY`. Jika konstanta tidak ditulis, yang akan dipakai adalah `PDO::FETCH_BOTH`.

Berikut contoh penggunaan dari `PDOStatement::fetchAll()`:

22.pdo_query_fetchall_num.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_NUM);
8 echo "<pre>";
9 print_r($arr);
10 echo "</pre>";
11
12 echo "<br>".$arr[2][1];

```

Hasil kode program:

PDO

```
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

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($arr)` di baris 9 memperlihatkan struktur dari array `$arr`. Sebagai contoh, untuk menampilkan isi kolom `nama_barang` (kolom ke-2) dari baris ke-3 perintahnya adalah `$arr[2][1]`. Jika kita ingin menampilkan semua nilai yang ada di dalam `$arr`, bisa memakai perulangan **foreach**, yang caranya sama seperti di pembahasan tentang **mysqli object**.

Berikut contoh penggunaan dari `PDOStatement::fetchAll(PDO::FETCH_ASSOC)`:

23.pdo_query_fetchall_assoc.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_ASSOC);
8 echo "<pre>";
9 print_r($arr);
10 echo "</pre>";
11
12 echo "<br>".$arr[2]["nama_barang"];
```

Hasil kode program:

PDO

```
Array
(
    [0] => Array
        (
            [id_barang] => 1
            [nama_barang] => TV Samsung 43NU7090 4K
            [jumlah_barang] => 5
            ...
        )
)
```

Laptop ASUS ROG GL503GE

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. Dengan demikian, untuk menampilkan isi kolom `nama_barang` dari baris ke-3 perintahnya adalah `$arr[2]["nama_barang"]`.

Selanjutnya, berikut contoh penggunaan dari `PDOStatement::fetchAll(PDO::FETCH_OBJ)`:

24.pdo_query_fetchall_obj.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_OBJ);
8 echo "<pre>";
9 print_r($arr);
10 echo "</pre>";
11
12 echo $arr[2]->nama_barang;
```

Hasil kode program:

```
Array
(
    [0] => stdClass Object
        (
            [id_barang] => 1
            [nama_barang] => TV Samsung 43NU7090 4K
            [jumlah_barang] => 5
            ...
        )
)
```

Laptop ASUS ROG GL503GE

Kembali, index dari dimensi pertama (untuk baris) tetap berbentuk angka. Namun dimensi kedua (untuk kolom) menggunakan penulisan object. Sehingga apabila ingin menampilkan isi kolom `nama_barang` dari baris ke-3 perintahnya adalah `$arr[2]->nama_barang`.

Konstanta berikutnya untuk method `fetchAll()` adalah PDO::FETCH_CLASS. Konstanta ini mirip seperti PDO::FETCH_OBJ, terutama jika dipanggil tanpa argument kedua. Maksudnya, kedua

pemanggilan ini akan menampilkan hasil yang sama:

```
13 ...
14 $arr = $stmt->fetchAll(PDO::FETCH_OBJ);
15 $arr = $stmt->fetchAll(PDO::FETCH_CLASS);
16 ...
```

Bedanya, jika menggunakan PDO::FETCH_CLASS, kita bisa menginput nama class sebagai argument kedua dari method `fetchAll()`.

Jika diperhatikan, object untuk setiap baris di dalam variabel `$arr` di-set oleh PHP sebagai **stdClass** object, yakni object "generik" bawaan PHP. Kita bisa mengatur agar PHP memakai class lain yang telah siapkan sebelumnya. Caranya, input nama class sebagai argument kedua dari method `fetchAll(PDO::FETCH_CLASS)`:

25.pdo_query_fetchall_obj_class.php

```
1 <?php
2 class MyClass{}
3
4 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
5 $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
6
7 $query = "SELECT * FROM barang";
8 $stmt = $pdo->query($query);
9
10 $arr = $stmt->fetchAll(PDO::FETCH_CLASS, "MyClass");
11 echo "<pre>";
12 print_r($arr);
13 echo "</pre>";
14
15 echo $arr[2]->nama_barang;
```

Hasil kode program:

```
Array
(
    [0] => MyClass Object
        (
            [id_barang] => 1
            [nama_barang] => TV Samsung 43NU7090 4K
            ...
        )
)
```

Laptop ASUS ROG GL503GE

Di baris 2 saya membuat class `MyClass` yang memang tidak berisi apa-apa. Class ini kemudian diinput sebagai argument kedua saat pemanggilan method `fetchAll()` di baris 10. Hasilnya, setiap baris sekarang merupakan *instance* dari class `MyClass`.

Cara seperti ini bisa dipakai untuk membuat teknik yang lebih rumit, misalnya mengisi class `MyClass` dengan property lain atau magic method `__set()` yang akan memproses nilai inputan.

Berikut contoh prakteknya:

26 pdo_query_fetchall_obj_class_set.php

```

1 <?php
2 class IlkoomBarang{
3     public $nama_toko = "Ilkoom Store";
4     public function __set($name, $value) {
5         $this->$name = strtoupper($value);
6     }
7 }
8
9 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
10
11 $query = "SELECT * FROM barang";
12 $stmt = $pdo->query($query);
13
14 $arr = $stmt->fetchAll(PDO::FETCH_CLASS, "IlkoomBarang");
15 echo "<pre>";
16 print_r($arr);
17 echo "</pre>";

```

Hasil kode program:

```

Array
(
    [0] => IlkoomBarang Object
        (
            [nama_toko] => Ilkoom Store
            [id_barang] => 1
            [nama_barang] => TV SAMSUNG 43NU7090 4K
            [jumlah_barang] => 5
            [harga_barang] => 5399000
            [tanggal_update] => 2019-01-17 15:02:47
        )

    [1] => IlkoomBarang Object
        (
            [nama_toko] => Ilkoom Store
            [id_barang] => 2
            [nama_barang] => KULKAS LG GC-A432HLHU

```

Di baris 2 - 7 saya membuat class IlkoomBarang. Class ini memiliki property \$nama_toko yang diisi dengan string "Ilkoom Store". Kemudian terdapat *magic method* __set() yang saya rancang agar setiap pengisian property yang tidak ada di dalam class, nilainya di proses dulu dengan fungsi strtoupper().

Pada saat pemanggilan method \$stmt->fetchAll(PDO::FETCH_CLASS, "IlkoomBarang") di baris 14, data dari tabel barang akan diinput ke dalam object IlkoomBarang. Buktiya, terdapat tambahan property [nama_toko] => Ilkoom Store di setiap object.

Selain itu karena terdapat magic method __set(), maka setiap nilai yang berasal dari tabel

barang akan di konversi menjadi huruf besar. Ini bisa dilihat dari isi property `nama_barang`. Untuk kolom lain tidak ada perubahan karena berisi angka.

Berikutnya, method `fetchAll()` juga bisa menerima inputan konstanta `PDO::FETCH_COLUMN`. Pilihan ini akan mengambil 1 kolom tabel, seperti contoh berikut:

27.pdo_query_fetchall_column.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT harga_barang FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_COLUMN);
8 echo "<pre>";
9 print_r($arr);
10 echo "</pre>";
11
12 echo $arr[2];

```

Hasil kode program:

```

Array
(
    [0] => 5399000
    [1] => 7600000
    [2] => 16200000
    [3] => 2099000
    [4] => 4750000
)
16200000

```

Karena yang diambil hanya 1 kolom, maka variabel `$arr` hanya berisi 1 dimensi saja. Jika ingin menampilkan data ketiga, tinggal akses `$arr[2]`.

Apabila hasil query `SELECT` mengembalikan lebih dari 1 kolom, maka kolom paling awal yang akan diambil oleh `fetchAll(PDO::FETCH_COLUMN)`:

28.pdo_query_fetchall_column_many.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_COLUMN);
8
9 echo "<pre>";
10 print_r($arr);
11 echo "</pre>";

```

Hasil kode program:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
```

Query yang dijalankan adalah "SELECT * FROM barang", artinya ambil semua nilai yang ada di tabel barang (5 baris dan 5 kolom). Namun karena kita menggunakan method `fetchAll(PDO::FETCH_COLUMN)`, maka yang diambil hanya kolom pertama saja, yakni kolom `id_barang` dalam contoh ini.

Konstanta `PDO::FETCH_COLUMN` juga bisa digabung dengan konstanta lain untuk proses filter data, misalnya seperti ini:

```
1 ...
2 $arr = $stmt->fetchAll(PDO::FETCH_COLUMN | PDO::FETCH_UNIQUE);
3 ...
```

Tambahan `PDO::FETCH_UNIQUE` akan mengembalikan nilai kolom yang unik saja. Artinya, jika terdapat data yang berulang, hanya diambil 1 nilai. Ini kurang lebih sama seperti hasil penambahan clausa `DISTINCT` di perintah query MySQL.

Konstanta terakhir yang akan kita bahas untuk method `fetchAll()` adalah `PDO::FETCH_KEY_PAIR`. Ini dipakai untuk membuat pasangan key – value array dari 2 kolom tabel. Kolom pertama akan menjadi `key` dan kolom kedua berisi `value`. Berikut contoh penggunaannya:

29 pdo_query_fetchall_fetch_key_pair.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT id_barang,harga_barang FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_KEY_PAIR);
8 echo "<pre>";
9 print_r($arr);
10 echo "</pre>";
11
12 echo $arr[3];
```

Hasil kode program:

```
Array
(
```

PDO

```
[1] => 5399000
[2] => 7600000
[3] => 16200000
[4] => 2099000
[5] => 4750000
)
16200000
```

Query yang saya jalankan adalah "SELECT id_barang,harga_barang FROM barang". Hasilnya berbentuk 2 kolom data, yakni kolom **id_barang** dan **harga_barang**.

Karena konstanta yang dipakai adalah PDO::FETCH_KEY_PAIR, maka kolom **id_barang** akan menjadi key dari arrar \$arr, sedangkan kolom **harga_barang** menjadi nilai atau isi dari array tersebut.

Berikut contoh untuk kolom lain:

30 pdo_query_fetchall_fetch_key_pair_2.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT nama_barang,harga_barang FROM barang";
5 $stmt = $pdo->query($query);
6
7 $arr = $stmt->fetchAll(PDO::FETCH_KEY_PAIR);
8
9 echo "<pre>";
10 print_r($arr);
11 echo "</pre>";
12
13 echo $arr["TV Samsung 43NU7090 4K"]."<br>";
14 echo $arr["Kulkas LG GC-A432HLHU"]."<br>";
15 echo $arr["Printer Epson L220"]."<br>";
```

Hasil kode program:

```
Array
(
    [TV Samsung 43NU7090 4K] => 5399000
    [Kulkas LG GC-A432HLHU] => 7600000
    [Laptop ASUS ROG GL503GE] => 16200000
    [Printer Epson L220] => 2099000
    [Smartphone Xiaomi Pocophone F1] => 4750000
)
5399000
7600000
2099000
```

Sekarang pasangan kolom yang saya pilih adalah **nama_barang** dengan **harga_barang**. Hasilnya, dengan mengakses echo \$arr["TV Samsung 43NU7090 4K"] akan tampil harga barang untuk baris tersebut.

Penggunaan konstanta PDO::FETCH_KEY_PAIR hanya bisa dipakai untuk data yang hasilnya berbentuk 2 kolom, tidak boleh lebih atau kurang.

10.11. 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:

31.pdo_prepared_fetchall_num.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang WHERE id_barang = ?";
5 $stmt = $pdo->prepare($query);
6 $stmt->execute([4]);
7
8 $arr = $stmt->fetchAll(PDO::FETCH_NUM);
9 echo "<pre>";
10 print_r($arr);
11 echo "</pre>";
12
13 echo $arr[0][1];

```

Hasil kode program:

```

Array
(
    [0] => Array
        (
            [0] => 4
            [1] => Printer Epson L220
            [2] => 14
            [3] => 2099000
            [4] => 2019-01-17 15:02:47
        )
)

```

PDO

```
)  
)  
Printer Epson L220
```

Di baris 4 saya membuat query **SELECT** dengan kondisi **WHERE id_barang = ?**. Tanda tanya ini nantinya kita input secara terpisah.

Di baris 5, 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.

Pemanggilan method \$stmt->execute([4]) di baris 6 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 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].

Setelah proses execute, sisa kode program di baris 8 – 13 sama seperti pembahasan kita sebelumnya, dimana saya menampilkan isi query dengan method fetchAll(PDO::FETCH_NUM). Hasilnya, variabel \$arr berisi tabel barang dalam format *numeric array*. Isinya hanya ada 1 baris karena terdapat batasan kondisi WHERE id_barang = 4.

Bagaimana jika ada 2 kondisi? Tidak masalah, cukup tambahkan dua buah tanda tanya dan input 2 buah argument ke dalam method execute() seperti contoh berikut:

32 pdo_prepared_fetch_assoc.php

```
1 <?php  
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");  
3  
4 $query = "SELECT * FROM barang WHERE id_barang = ? OR  
5       nama_barang = ?";  
6 $stmt = $pdo->prepare($query);  
7 $stmt->execute([1, "Printer Epson L220"]);  
8  
9 while ($row = $stmt->fetch(PDO::FETCH_ASSOC)){  
10 echo $row['id_barang'];    echo " | ";  
11 echo $row['nama_barang']; echo " | ";  
12 echo $row['jumlah_barang']; echo " | ";  
13 echo $row['harga_barang']; echo " | ";  
14 echo $row['tanggal_update'];  
15 echo "<br>";  
16 }
```

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.

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.

Di dalam PDO, semua data inputan dianggap sebagai string. MySQL sendiri tidak akan komplain jika tipe data angka juga diinput sebagai string. Maksudnya, kedua query berikut bisa diproses sebagaimana mestinya:

```
SELECT harga_barang FROM barang WHERE id_barang = 2
SELECT harga_barang FROM barang WHERE id_barang = '2'
```

Di dalam MySQL, kolom `id_barang` saya set sebagai integer, namun MySQL tetap bisa memproses walaupun data untuk kolom tersebut diinput sebagai integer. PDO memanfaatkan hal ini sehingga kita tidak perlu mengatur tipe data setiap inputan.

Perbedaan lain adalah, proses **bind** di method `execute()` tidak harus berbentuk variabel, tapi bisa diisi dengan data langsung. Namun jika diinginkan, kita juga bisa mengisinya sebagai variabel seperti contoh berikut:

33.pdo_prepared_fetch_variable.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang WHERE id_barang = ? OR
5         nama_barang = ?";
6 $stmt = $pdo->prepare($query);
7
8 $id = 1;
9 $nama = "Printer Epson L220";
10 $stmt->execute([$id, $nama]);
11
12 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
13     echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
14     echo "<br>";
15 }
```

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

Query yang dipakai sama seperti sebelumnya, namun di baris 8 dan 9 saya menyiapkan variabel \$id dan \$nama untuk diinput ke dalam method `execute()` di baris 10. Selain itu memakai method `$stmt->fetch(PDO::FETCH_NUM)` untuk menampilkan data tabel.

10.12. Prepared Statement dengan Named Parameters

Sebelumnya kita memakai tanda tanya " ? " sebagai *placeholder* atau penanda inputan query untuk prepared statement. Di PDO, terdapat penulisan lain yang dikenal sebagai **named parameter**.

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*. Berikut contoh penggunaannya:

34.pdo_prepared_named_parameters.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang WHERE id_barang = :id OR
5         nama_barang = :nama";
6 $stmt = $pdo->prepare($query);
7
8 $stmt->execute(['id'=>1, 'nama'=>"Printer Epson L220"]);
9
10 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
11     echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
12     echo "<br>";
13 }
```

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
```

Perhatikan cara penulisan query di baris 4-5, saya memakai :id dan :nama sebagai pengganti dari tanda tanya " ? ", inilah cara penulisan *named parameters*. Pada saat proses **bind** menggunakan `execute()` di baris 8, argument method ditulis sebagai *associative array*, yakni berbentuk `['id'=>1, 'nama'=>"Printer Epson L220"]`.

Salah satu keuntungan dari penggunaan named parameter adalah kita tidak terikat dengan urutan *placeholder*. Selama nama index dalam *associative array* sesuai dengan nama *placeholder*, tidak pengaruh urutannya seperti contoh berikut:

35.pdo_prepared_named_parameters_2.php

```
1 <?php
```

```

2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang WHERE jumlah_barang < :jumlah OR
5     harga_barang > :harga";
6 $stmt = $pdo->prepare($query);
7
8 $stmt->execute(['harga'=>5000000, 'jumlah'=>15]);
9
10 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
11     echo $row[0]." | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
12     echo "<br>";
13 }

```

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

```

Pada saat penulisan query di baris 4 – 5, kondisi yang dipakai adalah "WHERE jumlah_barang < :jumlah OR harga_barang > :harga". Secara berurutan, penulisan placeholder-nya adalah :jumlah, lalu :barang. Namun di dalam argument method `execute()` ditulis sebagai `['harga'=>5000000, 'jumlah'=>15]`. Ini tidak jadi masalah karena patokan dari named parameter adalah nama, bukan posisi.

Ini berbeda jika menggunakan tanda tanya "?" dimana kita harus menulis argument sesuai urutan ketika menjalankan method `execute()`. Karena itu pula penggunaan tanda tanya ini disebut sebagai **positional placeholder** atau **positional parameters**.

Di dalam method `execute()`, named parameter ini kadang ditulis juga dengan menyertakan tanda titik dua ":" , tapi ini tidak wajib dan PHP Manual juga tidak mengatur tentang hal ini. Kita bisa menggunakan salah satu perintah berikut:

```

$stmt->execute(['harga'=>5000000, 'jumlah'=>15]);
$stmt->execute([':harga'=>5000000, ':jumlah'=>15]);

```

10.13. 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 praktekkan di versi **mysqli** object, dan berikut contohnya di dalam PDO:

36 pdo_prepared_multiple_execution.php

```

1 <?php
2
3 // Buat format tanggal hari ini
4 $sekarang = new DateTime('now', new DateTimeZone('Asia/Jakarta'));
5 $timestamp = $sekarang->format("Y-m-d H:i:s");
6
7 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
8
9 $query = "INSERT INTO barang (nama_barang, jumlah_barang,
10 harga_barang, tanggal_update) VALUES (:nama,:jumlah,:harga,:tanggal)";
11
12 $stmt = $pdo->prepare($query);
13
14 // Input data 1
15 $nama = "Cosmos CRJ-8229 - Rice Cooker";
16 $jumlah = 4;
17 $harga = 299000;
18 $tanggal = $timestamp;
19
20 $stmt->execute(['nama'=>$nama, 'jumlah'=>4, 'harga'=>$harga,
21 'tanggal'=>$tanggal]);
22 echo "Query Ok, ".$stmt->rowCount()." baris berhasil ditambah <br>";
23
24 // Input data 2
25 $arr_input = [
26 'nama' => "Philips Blender HR 2157",
27 'jumlah' => 11,
28 'harga' => 629000,
29 'tanggal' => $timestamp
30 ];
31
32 $stmt->execute($arr_input);
33 echo "Query Ok, ".$stmt->rowCount()." baris berhasil ditambah <br>";
34
35 echo "<hr>";
36
37 // Tampilkan data barang
38 $query = "SELECT * FROM barang";
39 $stmt = $pdo->query($query);
40
41 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
42 echo $row[0]." | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
43 echo "<br>";
44 }

```

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

```

Dalam kode program ini saya menambah 2 buah data baru ke dalam tabel `barang` dengan 1 penulisan query prepared.

Di awal program terdapat kode untuk menggenerate tanggal hari ini yang disimpan ke dalam variabel `$timestamp`. Kemudian di baris 9–10 adalah penulisan *prepared query* `INSERT`, dimana saya memakai *named parameter* untuk 4 data inputan.

Sebagai data pertama, saya membuat 5 variabel di baris 15 – 18, yang kemudian diinput ke dalam method `execute()` di baris 20.

Tanpa menulis ulang query, saya membuat data kedua dengan bentuk *associative array* `$arr_input` di baris 25 – 30, yang kemudian dipakai untuk pemanggilan method `execute()` di baris 32.

Di sini kita telah menjalankan 2 buah proses `execute()` untuk 1 query prepared. Sebagai latihan, anda bisa membuat kode program yang sama, tapi menggunakan *positioning parameter*, yakni memakai tanda tanya " ? " pada saat pembuatan query prepared.

10.14. Transaction Query dengan PDO

Untuk membuat transaction, PDO menyediakan 3 method:

- ◆ `PDO::beginTransaction()`
- ◆ `PDO::rollBack()`
- ◆ `PDO::commit()`

Perhatikan bahwa ketiga method ini "melekat" ke **PDO** object, bukan ke **PDOStatement** object.

Cara penggunaannya mirip seperti di versi **mysqli**, yakni kita membuka proses transaction dengan menjalankan method `PDO::beginTransaction()` lalu menulis query seperti biasa menggunakan prepared statement maupun tidak.

Jika terjadi masalah, semua query bisa dibatalkan dengan memanggil method `PDO::rollBack()`. Atau jika sudah oke, jalankan method `PDO::commit()` untuk mempermanenkan hasil query.

Berikut contoh kode program dari proses transaction di PDO:

Karena dalam contoh sebelumnya kita menambah data baru ke tabel `barang`, silahkan reset ulang dengan menjalankan file `bab09\20.mysql_generate.php`.

37.pdo_prepared_transaction.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $pdo->beginTransaction();
5
6 $query = "DELETE FROM barang WHERE id_barang = ?";
7 $stmt = $pdo->prepare($query);
8 $stmt->execute([2]);
9
10 $query = "DELETE FROM barang WHERE nama_barang = :nama";
11 $stmt = $pdo->prepare($query);
12 $stmt->execute(['nama'=>"TV Samsung 43NU7090 4K"]);
13
14 // Tampilkan isi tabel selama transaction
15 echo "<h3>Di dalam Transaction</h3>";
16 $query = "SELECT * FROM barang";
17 $stmt = $pdo->query($query);
18
19 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
20     echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
21     echo "<br>";
22 }
23
24 $pdo->rollBack(); // atau $pdo->commit()
25
26 echo "<hr>";
27
28 // Tampilkan isi tabel setelah transaction
29 echo "<h3>Setelah Transaction</h3>";
30 $query = "SELECT * FROM barang";
31 $stmt = $pdo->query($query);
32
33 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
34     echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
35     echo "<br>";
36 }

```

Hasil kode program:

Di dalam Transaction

3	Laptop ASUS ROG GL503GE	7	16200000	2019-01-21 08:31:33
4	Printer Epson L220	14	2099000	2019-01-21 08:31:33
5	Smartphone Xiaomi Pocophone F1	25	4750000	2019-01-21 08:31:33

Setelah Transaction

1	TV Samsung 43NU7090 4K	5	5399000	2019-01-21 08:31:33
2	Kulkas LG GC-A432HLHU	10	7600000	2019-01-21 08:31:33
3	Laptop ASUS ROG GL503GE	7	16200000	2019-01-21 08:31:33
4	Printer Epson L220	14	2099000	2019-01-21 08:31:33
5	Smartphone Xiaomi Pocophone F1	25	4750000	2019-01-21 08:31:33

Setelah pembuatan koneksi dengan database MySQL, di baris 4 saya membuka proses

transaction dengan memanggil method `$pdo->beginTransaction()`.

Kemudian di baris 6 – 12 terdapat query `DELETE` yang akan menghapus tabel barang dengan kondisi `id_barang = 2` dan `nama_barang = TV Samsung 43NU7090 4K`, hasilnya 2 baris data akan dihapus.

Namun di baris 24 saya menjalankan method `$pdo->rollBack()` yang akan membatalkan semua query sebelumnya. Sekarang, isi tabel barang sudah kembali seperti semula.

10.15. Proses Bind Manual untuk Prepared Statement

Dalam query MySQL, `LIMIT` adalah perintah tambahan yang berfungsi membatasi hasil data. Sebagai contoh, untuk menampilkan 3 barang dengan harga termurah, kita bisa memakai query:

```
SELECT * FROM barang ORDER BY harga_barang LIMIT 3
```

Namun PDO memiliki sebuah masalah ketika inputan `LIMIT` ini berasal dari prepared statement. Berikut contoh kasusnya:

38 pdo_prepared_limit_problem.php

```

1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5
6     $query = "SELECT * FROM barang ORDER BY harga_barang LIMIT :batas";
7     $stmt = $pdo->prepare($query);
8     $stmt->execute(['batas'=>3]);
9
10    while ($row = $stmt->fetch(PDO::FETCH_NUM)){
11        echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
12        echo "<br>";
13    }
14    $stmt = NULL;
15 }
16 catch (\PDOException $e) {
17     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
18 }
19 finally {
20     $pdo=NULL;
21 }
```

Hasil kode program:

```
Koneksi / Query bermasalah: SQLSTATE[42000]: Syntax error or access violation: 1064
You have an error in your SQL syntax; check the manual that corresponds to your
MariaDB server version for the right syntax to use near ''3'' at line 1 (42000)
```

Di sini saya menjalankan prepared statement dengan `LIMIT :batas`. Nilai `:batas` akan diinput dari perintah `$stmt->execute(['batas'=>3])`, namun hasilnya terjadi error.

Bisa di pastikan bahwa tidak ada yang salah dari penulisan query, hasil akhir yang diharapkan adalah "SELECT * FROM barang ORDER BY harga_barang LIMIT 3". Lalu dimana salahnya?

Ini berhubungan dengan cara PDO yang otomatis mengkonversi semua inputan prepared statement sebagai string. Akibatnya, kode di atas akan menghasilkan query sebagai berikut:

```
SELECT * FROM barang ORDER BY harga_barang LIMIT '3'
```

Perhatikan tambahan tanda kutip di bagian `LIMIT '3'`, inilah yang menyebabkan error. Untuk nilai batasan `LIMIT`, MySQL hanya bisa menerima nilai angka (integer), tidak bisa berbentuk string. Yang seharusnya dijalankan adalah `LIMIT 3`, bukan `LIMIT '3'`.

Jadi bagaimana solusinya? Ternyata PDO juga masih mengizinkan proses **bind** manual menggunakan method `PDOStatement::bindValue()` atau `PDOStatement::bindParam()`. Dengan memakai salah satu dari method ini, kita bisa mengatur tipe data nilai inputan.

Berikut contoh kode program dari penggunaan `PDOStatement::bindValue()`:

39.pdo_prepared_bind_value.php

```
1 <?php
2 try {
3     $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
4     $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5
6     $query = "SELECT * FROM barang ORDER BY harga_barang LIMIT :batas";
7     $stmt = $pdo->prepare($query);
8     $stmt->bindValue('batas', 3, PDO::PARAM_INT);
9     $stmt->execute();
10
11    while ($row = $stmt->fetch(PDO::FETCH_NUM)){
12        echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
13        echo "<br>";
14    }
15    $stmt = NULL;
16 }
17 catch (\PDOException $e) {
18     echo "Koneksi / Query bermasalah: ".$e->getMessage(). " (".$e->getCode()." )";
19 }
20 finally {
21     $pdo=NULL;
22 }
```

Hasil kode program:

4	Printer Epson L220	14	2099000	2019-01-21 08:31:33
5	Smartphone Xiaomi Pocophone F1	25	4750000	2019-01-21 08:31:33
1	TV Samsung 43NU7090 4K	5	5399000	2019-01-21 08:31:33

Perbedaan dengan contoh sebelumnya hanya di baris 8 dan 9. Di baris 8 terdapat pemanggilan method `bindValue()`. Method ini butuh 3 buah argument:

1. Named parameter, yang dalam contoh ini berupa string 'batas'.
2. Nilai untuk named parameter, dalam contoh ini adalah angka 3.
3. Konstanta untuk tipe data dari named parameter. Karena angka 3 ingin dikirim sebagai integer, maka konstanta-nya adalah `PDO::PARAM_INT`.

Untuk parameter ke-3 ini, konstanta tipe data yang tersedia adalah:

- ◆ `PDO::PARAM_BOOL`, untuk tipe data boolean.
- ◆ `PDO::PARAM_NULL`, untuk nilai NULL.
- ◆ `PDO::PARAM_INT`, untuk tipe data integer.
- ◆ `PDO::PARAM_STR`, untuk tipe data string, char, atau varchar.
- ◆ `PDO::PARAM_LOB`, untuk tipe data blob.

Jika argument ketiga dari method `bindValue()` tidak diisi, nilai default adalah `PDO::PARAM_STR` (dianggap sebagai string).

Dengan demikian, perintah `$stmt->bindValue('batas', 3, PDO::PARAM_INT)` bisa dibaca: "tukar named parameter batas dengan angka 3 dan kirim ke MySQL sebagai tipe data integer". Hasilnya, query `SELECT...LIMIT` bisa diproses sebagaimana mestinya.

Di baris 8, pemanggilan method `execute()` tidak butuh lagi tambahan argument karena proses **bind** sudah kita lakukan terpisah.

Pemanggilan method `bindValue()` ini tidak untuk `LIMIT` saja, tapi juga bisa dipakai mengisi nilai prepared statement biasa. Jika terdapat lebih dari 1 nilai, method `bindValue()` harus dipanggil beberapa kali (satu untuk setiap nilai) seperti contoh berikut:

40.pdo_prepared_bind_value_2.php

```

1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3 $query = "SELECT * FROM barang WHERE id_barang = :id OR
4           nama_barang = :barang";
5
6 $stmt = $pdo->prepare($query);
7 $stmt->bindValue('id', 5, PDO::PARAM_INT);
8 $stmt->bindValue('barang', "Printer Epson L220", PDO::PARAM_STR);
9 $stmt->execute();
10
11 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
12   echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
13   echo "<br>";
14 }
```

Hasil kode program:

```
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-21 08:31:33
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-21 08:31:33
```

Di baris 7-8 terdapat 2 kali pemanggilan method `bindValue()`, karena di dalam query juga butuh 2 buah nilai inputan, yakni `:id` dan `:barang`.

Dalam 2 contoh ini saya memakai penulisan *named parameter*, bagaimana jika prepared statement di tulis memakai *positional parameters*, yakni memakai tanda tanya ?

Untuk *positional parameters*, argument pertama dari method `bindValue()` diisi dengan urutan posisi tanda "?" dimulai dari angka 1 untuk tanda tanya pertama, angka 2 untuk tanda tanya kedua, dst. Jika menggunakan *positional parameters*, kode program sebelumnya bisa ditulis sebagai berikut:

41.pdo_prepared_bind_value_positional.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3 $query = "SELECT * FROM barang WHERE id_barang = ? OR
4           nama_barang = ?";
5
6 $stmt = $pdo->prepare($query);
7 $stmt->bindValue(1, 5, PDO::PARAM_INT);
8 $stmt->bindValue(2, "Printer Epson L220", PDO::PARAM_STR);
9 $stmt->execute();
```

Karena terdapat 2 buah tanda tanya, maka argument pertama dari method `bindValue()` ditulis sebagai 1 dan 2, sesuai dengan urutan tanda tanya yang ada di query.

Alternatif method untuk membuat proses bind secara manual adalah `PDOStatement::bindParam()`. Method ini juga butuh 3 buah argument sebagaimana method `bindValue()`, namun untuk `bindParam()`, nilai harus diisi dalam bentuk variabel, tidak bisa berisi angka langsung.

Berikut contoh kode program dari penggunaan `PDOStatement::bindParam()`:

42.pdo_prepared_bind_param.php

```
1 <?php
2 $pdo = new PDO("mysql:host=localhost;dbname=ilkoom", "root", "");
3
4 $query = "SELECT * FROM barang ORDER BY harga_barang LIMIT :batas";
5 $stmt = $pdo->prepare($query);
6 $stmt->bindParam('batas', $batas, PDO::PARAM_INT);
7 $batas=3;
8 $stmt->execute();
9
10 while ($row = $stmt->fetch(PDO::FETCH_NUM)){
11   echo $row[0]. " | ".$row[1]. " | ".$row[2]. " | ".$row[3]. " | ".$row[4];
```

```
12 echo "<br>";
13 }
```

Hasil kode program:

```
4 | Printer Epson L220 | 14 | 2099000 | 2019-01-21 08:31:33
5 | Smartphone Xiaomi Pocophone F1 | 25 | 4750000 | 2019-01-21 08:31:33
1 | TV Samsung 43NU7090 4K | 5 | 5399000 | 2019-01-21 08:31:33
```

Di sini saya kembali memakai query "LIMIT :batas". Perhatikan cara pemanggilan method `bindParam()` di baris 6. Sebagai argument kedua, kita tidak bisa langsung menginput angka 3 seperti di `bindValue()`, tapi harus diisi dengan variabel. Dalam contoh ini saya menggunakan variabel `$batas` yang isinya diinput di baris 7. Jika ditulis langsung sebagai `bindParam('batas', 3, PDO::PARAM_INT)` maka akan menghasilkan error.

Di dalam PDO, proses bind manual seperti ini memang bukan sebuah keharusan. Dalam banyak hal, menginput langsung argument ke dalam method `execute()` lebih praktis daripada memanggil method `bindValue()` atau `bindParam()`. Pengecualiannya untuk kasus-kasus khusus seperti LIMIT, atau inputan yang sensitif terhadap tipe data.

Exercise

Sebagai penutup serta menguji pemahaman tentang PDO (dan juga perintah query MySQL), saya tantang anda membuat kode program untuk meng-generate tabel **mahasiswa** serta menampilkan isinya.

Latihan ini mirip seperti kode program yang dipakai untuk me-reset tabel **barang**. Dimana jika kode ini dijalankan kembali, tabel **mahasiswa** otomatis di reset ulang. Urutan kode yang perlu dibuat adalah:

1. Buat koneksi dengan MySQL menggunakan PDO.
2. Buat database iloom 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).

Berikut tampilan dan isi dari tabel mahasiswa:

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

Isi tabel mahasiswa

Tipe data data untuk setiap kolom boleh bebas, yang penting semua data bisa tersimpan ke database. Khusus untuk kolom tanggal lahir, sesuaikan dengan format tipe data DATE dari MySQL, yakni dengan format yyyy-mm-dd.

Tips: karena dalam kode program ini perlu membuat database `ilkoom` dan menggunakan, maka pada saat koneksi (pembuatan PDO object), kita tidak bisa langsung menggunakan database. Setelah database `ilkoom` di buat, baru pilih dengan perintah query "USE `ilkoom`".

Tips lagi: Silahkan lihat kode program yang dipakai untuk me-reset tabel `barang` sebagai panduan (file `bab09\20.mysql_generate.php`).

Berikut tampilan akhir ketika file dijalankan:

```

Database 'ilkoom' berhasil di buat / sudah tersedia
Database 'ilkoom' berhasil di pilih
Tabel 'mahasiswa' berhasil di buat
Tabel 'mahasiswa' berhasil di isi 5 baris data

Tabel Mahasiswa

13012012 | James Situmorang | Medan | 1995-04-02 | Kedokteran
14005011 | Riana Putria | 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

```

Proses pembuatan (reset) tabel mahasiswa

Selamat jika anda berhasil!

Apabila butuh untuk menyamakan kode program (atau menyerah), silahkan buka file `bab_10\43 pdo_exercise.php`.