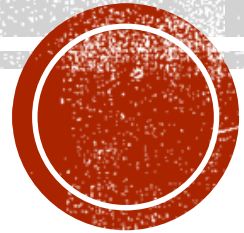
Yii2: Working with Databases

Oleh: Ahmad Syauqi Ahsan



Yii DAO (Database Access Object)

- Yii DAO dibangun diatas PDO (PHP Data Objects).
  - PDO merupakan extension yang menyediakan antar muka untuk mengakses database menggunakan bahasa PHP
- Yii DAO menyediakan object-oriented API untuk mengakses database relasional.
- Yii DAO merupakan dasar untuk metode pengaksesan database secara lebih advanced seperti Query Builder dan Active Record
- Yii DAO dapat langsung mendukung DBMS berikut ini:
  - MySQL
  - MariaDB
  - SQLite
  - PostgreSQL
  - CUBRID versi 9.3 keatas
  - Oracle
  - MSSQL versi 2008 keatas

Koneksi ke Database

```
return [
    // ...
    'components' => [
        // ...
        'db' => [
            'class' => 'yii\db\Connection',
            'dsn' => 'mysql:host=localhost;dbname=dbku',
            'username' => 'userku',
            'password' => 'passku',
            'charset' => 'utf8',
        ],
    ],
    // ...
];
```

- Untuk dapat mengakses database, anda harus terhubung ke database tersebut dengan cara membuat instance dari yii\db\Connection
- Akses ke database seringkali dilakukan dari beberapa tempat berbeda → koneksi database akan lebih tepat jika dikonfigurasikan pada application component (lihat script diatas).
- Anda dapat mengakses koneksi ke DB diatas melalui **`Yii::$app->db`**
- Konfigurasi diatas merupakan contoh koneksi ke database MySQL. Untuk database lain, pastikan nilai dari property **`'dsn'`** diisi dengan nilai yang sesuai dengan database server yang digunakan.
  - Sebagai contoh, nilai property **`'dsn'`** untuk database PostgreSQL adalah: **`'pgsql:host=localhost;port=5432;dbname=mydatabase'`**

Menjalankan SQL Query

- Untuk menjalankan SQL Query → panggil fungsi createCommand dan memasukkan SQL Query sebagai parameternya. Kemudian panggil salah satu dari beberapa metode eksekusi yang tersedia (lihat script disamping).

- Catatan: untuk menjaga presisi, data yang diambil dari database selalu direpresentasikan sebagai ***string*** (teks), walaupun tipe data dari kolom pada table adalah numerik.

```
$db = Yii::$app->db;

// return a set of rows. each row is an
// associative array of column names and values.
// an empty array is returned if no results
$posts = $db->createCommand('SELECT * FROM post')
->queryAll();

// return a single row (the first row)
// false is returned if no results
$post = $db->createCommand('SELECT * FROM post
WHERE id=1')->queryOne();

// return a single column (the first column)
// an empty array is returned if no results
$titles = $db->createCommand('SELECT title FROM
post')->queryColumn();

// return a scalar
// false is returned if no results
$count = $db->createCommand('SELECT COUNT(*) FROM
post')->queryScalar();
```

Binding Parameters

- Untuk menghindari serangan SQL Injection, anda harus menggunakan pendekatan ***binding parameters***.
- Pendekatan ini juga dapat meningkatkan performa dari statemen SQL.
- Anda dapat menggunakan metode:
  - `bindValue()`: bind a single parameter value
  - `bindValues()`: bind multiple parameter value in one call
  - `bindParam()`: similar to `bindValue()` but also support binding parameter references
- Lihat contoh disamping.

```
$db = Yii::$app->db;

$post = $db->createCommand('SELECT * FROM post
WHERE id=:id AND status=:status')
->bindValue(':id', $_GET['id'])
->bindValue(':status', 1)
->queryOne();

// ..
$params = [':id' => $_GET['id'], ':status' => 1];

$post = $db->createCommand('SELECT * FROM post
WHERE id=:id AND status=:status')
->bindValues($params)
->queryOne();
// OR
$post = $db->createCommand('SELECT * FROM post
WHERE id=:id AND status=:status', $params)
->queryOne();

// ..
$command = $db->createCommand('SELECT * FROM post
WHERE id=:id')->bindParam(':id', $id);
$id = 1;
$post1 = $command->queryOne();
$id = 2;
$post2 = $command->queryOne();
```

Menjalankan perintah SQL selain Query

- Untuk menjalankan perintah SQL selain query dapat dilakukan dengan memanggil metode **`execute()`**.
- Untuk perintah-perintah DML, selain menuliskan statemen SQL dasar, dapat juga dilakukan dengan memanggil metode **`insert()`**, **`update()`**, atau **`delete()`**, sebelum memanggil metode **`execute()`**.

```
$db = Yii::$app->db;

$db->createCommand('UPDATE post SET status=1 WHERE
id=1')->execute();

// INSERT (table name, column values)
$db->createCommand()->insert('user', ['name' =>
'Sam', 'age' => 30,])->execute();

// UPDATE (table name, column values, condition)
$db->createCommand()->update('user', ['status' =>
1], 'age > 30')->execute();

// DELETE (table name, condition)
$db->createCommand()->delete('user', 'status =
0')->execute();

// BATCH INSERT
$db->createCommand()->batchInsert('user', ['name',
'age'], [['Tom', 30], ['Jane', 20], ['Linda', 25],
])->execute();
```

Meletakkan nama table dan kolom dalam tanda petik (Quote)

- Ada kalanya anda perlu menuliskan perintah SQL yang didalamnya terdapat nama table atau kolom dalam tanda petik.
- Misal anda ingin menjalankan perintah **`SQL SELECT SUM(‘SALARY’) FROM EMPLOYEE`**
- Aturan penggunaan quote berbeda antara DBMS satu dengan lainnya → menambah kerumitan dalam menulis perintah SQL untuk aplikasi yang database-agnostic (tidak terikat dengan jenis DBMS)
- Dalam Yii DAO, anda dapat meng-quote nama kolom dalam kurung kotak “[ ]” dan nama table dalam kurung kurawal “{ }”. Lihat contoh disamping.

```
$db = Yii::$app->db;

// executes this SQL for MySQL: SELECT COUNT(`id`) FROM `employee`
$count = $db->createCommand("SELECT COUNT([[id]]) FROM {{employee}}")->queryScalar();
```

Multiple DBMS

- Dalam system yang besar/komplek, seringkali terdapat beberapa server database untuk data yang sama.
- Untuk performa, anda mungkin perlu mengaplikasikan strategi bahwa penulisan data dilakukan di DBMS master sedangkan pembacaan data di DBMS slave (read-write splitting).
- Untuk menerapkan strategi diatas, anda perlu mengatur konfigurasi koneksi database anda seperti script disamping ini.
- Dengan konfigurasi seperti script disamping, setiap penggunaan metode `execute()` akan dianggap sebagai perintah ***write*** dan dilakukan pada DBMS master.
- Sedangkan penggunaan metode `query`***`XYZ`***`()` akan dianggap sebagai perintah ***read*** dan dilakukan pada DBMS slave.

```
[
  'class' => 'yii\db\Connection',

  // configuration for the master
  'dsn' => 'dsn for master server',
  'username' => 'master',
  'password' => 'masterPassword',

  // common configuration for slaves
  'slaveConfig' => [
    'username' => 'slave',
    'password' => 'slavePassword',
    'attributes' => [
      // use a smaller connection timeout
      PDO::ATTR_TIMEOUT => 10,
    ],
  ],

  // list of slave configurations
  'slaves' => [
    ['dsn' => 'dsn for slave server 1'],
    ['dsn' => 'dsn for slave server 2'],
    ['dsn' => 'dsn for slave server 3'],
    ['dsn' => 'dsn for slave server 4'],
  ],
]
```

Bekerja dengan Database Schema

Yii DAO menyediakan metode lengkap untuk memanipulasi objek-objek didalam database

Metode-metode tersebut adalah:

- **`createTable()`**: creating a table
- **`renameTable()`**: renaming a table
- **`dropTable()`**: removing a table
- **`truncateTable()`**: removing all rows in a table
- **`addColumn()`**: adding a column
- **`renameColumn()`**: renaming a column
- **`dropColumn()`**: removing a column
- **`alterColumn()`**: altering a column
- **`addPrimaryKey()`**: adding a primary key
- **`dropPrimaryKey()`**: removing a primary key
- **`addForeignKey()`**: adding a foreign key
- **`dropForeignKey()`**: removing a foreign key
- **`createIndex()`**: creating an index
- **`dropIndex()`**: removing an index

Query Builder

```
$query = new \yii\db\Query();
$rows = $query->select(['id', 'email'])
  ->from('user')
  ->where(['last_name' => 'Jono'])
  ->limit(10)
  ->all();
```

- Query Builder dibangun diatas Yii DAO
- Dengan Query Builder memungkinkan anda untuk membentuk statemen SQL secara ***agnostic***.
- Penggunaan Query Builder normalnya membutuhkan 2 langkah:
  1. Membuat objek yii\db\Query yang merepresentasikan SQL Query
  2. Menjalankan metode query (misal: `all()`) dari yii\db\Query untuk mengambil data dari database.
- Script diatas akan menjalankan statemen SQL seperti dibawah ini:

```
SELECT `id`, `email`
FROM `user`
WHERE `last_name = :last_name
LIMIT 10
```

- Dimana bind parameter **:last_name** diisi dengan teks **"Jono"**
- Catatan: Query Builder secara otomatis akan menggunakan bind parameter.

select()

- Metode `select()` digunakan untuk memilih kolom, misal:
```
$query->select(['id', 'email']);
```
- Dalam SQL Query, seringkali kita menambahkan nama table didepan nama kolom yang dipilih. Anda dapat melakukannya dengan:
```
$query->select(['user.id', 'user.email']);
```
- Anda juga dapat menambahkan alias untuk kolom dengan cara:
```
$query->select(['id AS id_user', 'email']);
```
atau
```
$query->select(['id' => 'id_user' , 'email']);
```
- Anda juga dapat menggunakan fungsi dalam database dengan cara:
```
$query->select([["CONCAT(first_name, ' ', last_name) AS full_name",
'email']);
```
- Anda dapat menggunakan metode andSelect() untuk menambahkan kolom yang dipilih:
```
$query->select(['id', 'email']);
$query->addSelect(['last_name']);
```

from()

- Metode `from()` digunakan untuk menentukan table dimana query akan dilakukan, misal:
  ```
  $query->from('user');
  ```
- Anda juga dapat menjalankan SQL Query pada lebih dari satu table. Contoh:
  ```
  $query->from(['user', 'post'])
  ```
- Untuk menggunakan table alias dapat anda lakukan dengan cara:
  ```
  $query->from(['user u', 'post p'])
  ```
  atau
  ```
  $query->from(['u' => 'user', 'p' => 'post'])
  ```
- Anda juga dapat menggunakan sub query pada `from()` . Contoh:
  ```
  $subQuery = (new Query())->select('id')->from('user')
  ->where('status = 1');
  // SELECT * FROM (SELECT `id` FROM `user` WHERE status=1) u
  $query->from(['u' => $subQuery]);
  ```

where()

- Metode **where()** digunakan untuk menuliskan kondisi dari SQL Query. Misal:
```
$query->where('status = 1');
```
- Atau dengan menggunakan bind parameter:
```
$query->where('status = :status', [':status' => $status]);
```
- **JANGAN** memasukkan variable didalam **where()** secara langsung Karena akan rentan diserang dengan SQL Injection.
Berikut ini contoh yang **HARUS** dihindari:
```
$query->where('status = $status');
```
- Anda dapat menuliskan beberapa kondisi didalam **where()** menggunakan *Hash Format*. Beberapa kondisi ini digabungkan dengan operator **'AND'**. Contoh:
```
$query->where(['status' => 10, 'type' => null, 'id' => [4, 8, 15]]);
```
- Anda juga dapat menggunakan sub query pada **where()** . Contoh:
```
$userQuery = (new Query())->select('id')->from('user');
// ... WHERE 'id' IN (SELECT 'id' FROM 'user')
$query->where(['id' => $userQuery]);
```

andWhere() dan orWhere()

- Metode **andWhere()** dan **orWhere()** digunakan untuk menambahkan kondisi didalam sebuah query. Misal:

```
$status = 10;
$search = 'yii';

$query->where(['status' => $status]);
if (!empty($search)) {
  $query->andWhere(['like', 'title', $search]);
}query->where(‘status = 1’);
```

- Jika variable $search tidak kosong, SQL Query berikut akan di-generate:

```
... WHERE (‘status’ = 10) AND (‘title’ LIKE ‘%yii%’)
```

filterWhere()

- Ketika membuat kondisi pada WHERE yang melibatkan masukan dari user, biasanya anda ingin mengabaikan jika nilai inputnya kosong.
- Misal pencarian berdasarkan username dan email, namun akan mengabaikan kondisi jika username atau email kosong, dapat ditulis dengan:

```
// $username dan $email diambil dari masukan user
$query->filterWhere(['username' => $username, 'email' => $email]);
```

- Jika variable $email kosong sedangkan $username ada nilainya, maka query yang akan di-generate adalah::

```
... WHERE ('username' = :username)
```

- Mirip seperti metode `andWhere()` dan `orWhere()`, anda juga dapat menggunakan metode `andFilterWhere()` dan `orFilterWhere()` untuk menambahkan kondisi didalam sebuah query.

orderBy(), groupBy(), having()

- Pengurutan. `orderBy()` digunakan untuk menentukan bagian `ORDER BY` pada sebuah SQL. Contoh:
  `$query->orderBy(['id' => SORT_ASC, 'name' => SORT_DESC]);`
- Anda juga bias menambahkan kolom pada bagian `ORDER BY` pada sebuah SQL dengan memanggil metode `addOrderBy()`.
- Pengelompokan berdasarkan kolom. `groupBy()` digunakan untuk menentukan bagian `GROUP BY` pada sebuah SQL. Contoh:
  `$query->groupBy(['id', 'status']);`
- Anda juga bias menambahkan kolom pada bagian `GROUP BY` pada sebuah SQL dengan memanggil metode `addGroupBy()`.
- `having()` digunakan untuk menentukan bagian `HAVING` pada sebuah SQL. Contoh:
  `$query->having(['status' => 1]);`
- Anda juga bias menambahkan kolom pada bagian `HAVING` pada sebuah SQL dengan memanggil metode `addHaving()`.

`limit()` dan `offset()`

- **`limit()` dan `offset()`** digunakan untuk menentukan bagian **`LIMIT`** dan **`OFFSET`** pada sebuah SQL. Contoh:
  ```
  $query->limit(10)->offset(20);
  ```
- Untuk DBMS yang tidak mendukung LIMIT dan OFFSET (missal: MSSQL), query builder akan men-generate sattemen SQL yang mengemulasikan fungsi **`LIMIT`** atau **`OFFSET`**.

Query Methods

```
// SELECT `id`, `email` FROM `user`
$rows = (new \yii\db\Query())
->select(['id', 'email'])->from('user')
->all();

// SELECT * FROM `user` WHERE `username` LIKE
// `%test%`
$row = (new \yii\db\Query())->from('user')
->where(['like', 'username', 'test'])
->one();
```

- **all()**: returns an array of rows with each row being an associative array of name-value pairs.
- **one()**: returns the first row of the result.
- **column()**: returns the first column of the result.
- **scalar()**: returns a scalar value located at the first row and firstcolumn of the result.
- **exists()**: returns a value indicating whether the query contains any result.
- **count()**: returns the result of a COUNT query.
- Other aggregation query methods, including **sum($q)**, **average($q)**, **max($q)**, **min($q)**. The $q parameter is mandatory for these methods and can be either a column name or a DB expression.

Batch Query

- Ketika melakukan query pada table yang berisi banyak data , memanggil metode **`yii\db\Query::all()`** bukanlah pilihan yang tepat. Karena hal ini akan menggunakan memory yang besar.
- Batch query pada Yii akan memanfaatkan data cursor untuk mengambil data per bagian. (Perhatikan contoh disamping)
- Secara default, batch query akan mengambil 100 data untuk setiap iterasi.
- Anda dapat mengubahnya dengan mengisikan nilai ukuran sebagai parameter pertama ketika memanggil method batch(). Contoh: **`$query->batch(10);`**

```php
use yii\db\Query;

$query = (new Query())
->from('item');

foreach ($query->batch(10) as $i => $items) {
  // variable $items is an array of 10 or fewer
  // rows from the item table
  echo 'Batch ke: ' . $i . '<br/>';
  foreach ($items as $item) {
    echo $item['name'] . '<br/>';
  }
}

// or if you want to iterate the row one by one
foreach ($query->each() as $user) {
  // $user represents one row of data from the
  // user table
}
```

Active Record (AR)

- Active Record (AR) menyediakan antar muka secara objek-oriented untuk mengakses dan memanipulasi data pada database.
- Sebuah kelas AR diasosiasikan (dihubungkan) dengan sebuah table dalam database.
- Sebuah instance dari AR berkorespondensi dengan sebuah baris pada table tersebut.
- Misal Customer adalah sebuah AR yang diasosiasikan dengan table "customer". Maka, untuk menambahkan sebuah baris dalam table customer dapat dilakukan dengan cara:

```
$customer = new Customer();
$customer->name = 'Qiang Xue';
$customer->save();
```

- Script diatas sama dengan menggunakan Query biasa seperti berikut:

```
$db->createCommand('INSERT INTO `customer` (`name`) VALUES
(:name)', [':name' => 'Qiang Xue'])->execute();
```

Keuntungan Menggunakan AR

- Keuntungan menggunakan AR adalah:
  - Lebih mudah dipahami
  - Tidak rawan terjadi kesalahan penulisan
  - Tidak ada masalah kompatibilitas jika menggunakan beberapa database yang berbeda
- AR pada Yii mendukung RDBMS berikut ini:
  - MySQL 4.1 or later: via yii\db\ActiveRecord
  - PostgreSQL 7.3 or later: via yii\db\ActiveRecord
  - SQLite 2 and 3: via yii\db\ActiveRecord
  - Microsoft SQL Server 2008 or later: via yii\db\ActiveRecord
  - Oracle: via yii\db\ActiveRecord
  - CUBRID 9.3 or later: via yii\db\ActiveRecord
  - Sphinx: via yii\sphinx\ActiveRecord, requires the yii2-sphinx extension
  - ElasticSearch: via yii\elasticsearch\ActiveRecord, requires the yii2-elasticsearch extension
- AR pada Yii juga mendukung NoSQL database berikut ini:
  - Redis 2.6.12 or later: via yii\redis\ActiveRecord, requires the yii2-redis extension
  - MongoDB 1.3.0 or later: via yii\mongodb\ActiveRecord, requires the yii2-mongodb extension

Deklarasi AR Class

- Deklarasi AR dilakukan dengan cara membuat class yang meng-extend yii\db\ActiveRecord
- Dalam class ini method **`tableName()`** harus di-override untuk menentukan table yang diasosiasikan dengan class ini
- Instance AR dianggap sebagai Model. Sehingga class AR biasanya diletakkan di dalam namespace app\models
- Yii\db\ActiveRecord merupakan class yang meng-extend class yii\base\Model, sehingga semua fitur dari sebuah class Model juga dimiliki oleh class AR (attributes, validation rules, dll)
- Secara default, class AR akan menggunakan komponen aplikasi **"db" (Yii::$app->db)** untuk terhubung dengan database.

```php
/**
* Berikut ini merupakan class AR dengan nama
* Customer yang diasosiasikan dengan tabel
* customer
*/

namespace app\models;

use yii\db\ActiveRecord;

class Customer extends ActiveRecord
{
  const STATUS_INACTIVE = 0;
  const STATUS_ACTIVE = 1;

  /**
  * @return string the name of the table
  * associated with this ActiveRecord class.
  */
  public static function tableName()
  {
    return 'customer';
  }
}
```

Query Data

- Setelah AR terdeklarasi, anda dapat menggunakannya untuk meng-query data dengan langkah2 seperti berikut:
  1. Buat sebuah objek query dengan memanggil metode **`yii\db\ActiveRecord::find()`**
  2. Atur objek query dengan memanggil metode2 query building (seperti: **`where()`**, **`orderBy()`**, dll)
  3. Panggil salah satu metode query untuk mengambil data (seperti: **`all()`**, **`one()`**, dll)
- Cara diatas mirip seperti penggunaan Query Builder, hanya saja output yang dihasilkan berbentuk objek AR

```
// return a single customer whose ID is 123
// SELECT * FROM `customer` WHERE `id` = 123
$customer = Customer::find()  // Langkah 1
->where(['id' => 123])        // Langkah 2
->one();                      // Langkah 3

// return all active customers and sor by IDs
// SELECT * FROM `customer` WHERE `status` = 1
// ORDER BY `id`

$customers = Customer::find()
->where(['status' => Customer::STATUS_ACTIVE])
->orderBy('id')
->all();

// return the number of active customers
// SELECT COUNT(*) FROM `customer` WHERE `status`
// = 1
$count = Customer::find()
->where(['status' => Customer::STATUS_ACTIVE])
->count();

// return all active customers in an array indexed
// by customer IDs
// SELECT * FROM `customer`
$customers = Customer::find()
->indexBy('id')->all();
```

`findOne()` & `findAll()`

- Metode **`findOne()`** dan **`findAll()`** ini merupakan shortcut yang disediakan Yii untuk melakukan query berdasarkan nilai primary key atau nilai dari beberapa kolom.
- Metode **`findOne()`** ini akan menghasilkan 1 baris data pertama hasil dari query.
- Metode **`findAll()`** ini akan menghasilkan seluruh baris data hasil dari query.
- Kedua metode ini dapat menerima parameter berikut ini:
  - Nilai scalar value: nilai tersebut dianggap sebagai primary key untuk data yang akan dicari.
  - Sebuah array berisi nilai scalar: nilai-nilai tersebut diperlakukan sebagai nilai primary key yang akan dicari.
  - Sebuah associative array (key-value pair):

```php
// SELECT * FROM `customer` WHERE `id` = 123
// returns a single customer whose ID is 123
$customer = Customer::findOne(123);

// returns customers whose ID is 100, 101, or 123
// SELECT * FROM `customer` WHERE `id` IN (100,
// 101, 123)
$customers = Customer::findAll([100, 101, 123]);

// returns an active customer whose ID is 123
// SELECT * FROM `customer` WHERE `id` = 123 AND
// `status` = 1
$customer = Customer::findOne(['id' => 123,
  'status' => Customer::STATUS_ACTIVE,
]);

// returns all inactive customers
// SELECT * FROM `customer` WHERE `status` = 0
$customer = Customer::findAll([
  'status' => Customer::STATUS_INACTIVE,
]);
```

Mengakses data pada AR

- Query melalui class AR akan menghasilkan data dalam bentuk objek atau instance AR.
- Setiap atribut dari objek AR memiliki nama sesuai dengan nama kolom pada table yang diasosiasikan dengan class AR (case-sensitive).
- Atribut2 tersebut di-generate oleh Yii secara otomatis. Jangan membuat atribut baru pada class AR dengan nama yang sama dengan nama kolom pada table yang diasosiasikan dengan class AR tersebut.

```
// "id" and "email" are the names of columns in the "customer" table
$customer = Customer::findOne(123);
$id = $customer->id;
$email = $customer->email;
```

Transformasi Data

- Seringkali data akan ditampilkan dengan format berbeda dari ketika data tersebut disimpan dalam database
- Misalkan anda ingin menampilkan kolom birthday, yang disimpan di database dalam format timestamp, menggunakan format ‘YYYY/MM/DD”.
- Anda dapat melakukannya dengan menambahkan metode yang berfungsi untuk transformasi data tersebut. Lihat kode disamping.

```
class Customer extends ActiveRecord
{
  // ...

  // ingat metode getter() dan setter()!
  public function getBirthdayText()
  {
    return date('Y/m/d', $this->birthday);
  }

  public function setBirthdayText($value)
  {
    $this->birthday = strtotime($value);
  }
}

// ----------------------------------------//

// Sekarang anda dapat mengakses kolom birthday
// dengan cara:

$customer = Customer::findOne(123);
$id = $customer->id;
$birthday = $customer->birthdayText;
```

Mengambil Data dalam bentuk Array

- Mengambil data dalam bentuk objek AR sangat *convenient*, namun membutuhkan alokasi memory yang cukup besar.
- Untuk menghemat penggunaan memory, anda dapat mengambil data dalam bentuk array, dengan cara memanggil metode **`asArray()`** sebelum menjalankan metode query.
- Kekurangan dari metode ini adalah anda akan kehilangan fitur-fitur yang melekat pada objek AR.
  Misalnya: tipe data dari atribut pada objek AR otomatis disesuaikan dengan tipe data pada kolom di table. Sedangkan pada array, semua data bertipe string.

```
// return all customers
// each customer is returned as an associative array
// where the indexes are column names in table
$customers = Customer::find()
->asArray()
->all();
```

Mengambil Data dalam Batch

- Anda juga dapat mengambil data per kelompok (batch).
- Cara ini dapat menghemat penggunaan memory, khususnya pada table yang memiliki banyak baris data.

```
// fetch 10 customers at a time
foreach (Customer::find()->batch(10) as $customers) {
  // $customers is an array of 10 or fewer Customer objects
}

// fetch 10 customers at a time and iterate them one by one
foreach (Customer::find()->each() as $customer) {
  // $customer is a Customer object
}

// batch query with eager loading
foreach (Customer::find()->with('orders')->each() as $customer) {
  // $customer is a Customer object
}
```

Menyimpan Data

- Dengan AR, anda dapat dengan mudah menyimpan data kedalam database:
  - Siapkan instance AR
  - Berikan nilai untuk attribute pada AR
  - Panggil metode **`yii\db\ActiveRecord::save()`** untuk menyimpan data kedalam database
- Metode **`save()`** dapat melakukan INSERT ataupun UPDATE, tergantung keadaan instance AR. (Lihat script disamping)

```php
// insert a new row of data
$customer = new Customer();
$customer->name = 'James';
$customer->email = 'james@example.com';
$customer->save();

// update an existing row of data
$customer = Customer::findOne(123);
$customer->email = 'james@newexample.com';
$customer->save();
```

Validasi Data

- Karena class `yii\db\ActiveRecord` di-extend dari class `yii\base\Model` → class AR juga memiliki fitur validasi yang sama.
- Anda dapat mendeklarasikan aturan validasi pada class AR dengan meng-override fungsi `rules()`.
- Ketika anda memanggil metode `save()`, secara default dia akan menjalankan metode `validate()` terlebih dahulu.
- Jika metode `validate()` mengembalikan nilai true maka data akan disimpan ke dalam database.
- Sebaliknya, jika terdapat kesalahan validasi, maka metode `validate()` akan mengembalikan nilai true. Dan data tidak akan disimpan ke dalam database.
- Jika anda ingin menyimpan data tanpa menjalankan metode `validate()`, anda dapat memanggil fungsi `save(false)`.

```
class Customer extends ActiveRecord
{
  // ...

  public function rules() {
    return [
      [['name'], 'required'],
      [['name'], 'string'],
      [['email'], 'string', 'max' => 150],
    ];
  }
}

// ----------------------------------------//

// insert a new row of data with validation
$customer = new Customer();
$customer->name = 'James';
$customer->email = 'james@example.com';
$customer->save();

// insert a new row of data without validation
$customer = new Customer();
$customer->name = 'Jono';
$customer->email = 'jono@example.com';
$customer->save(false);
```

Dirty Attributes

- Ketika anda memanggil metode **`save()`** untuk menyimpan/meng-update data melalui AR, Yii hanya akan menyimpan ***dirty attributes*** saja.
- Sebuah attribute dianggap ***dirty*** jika nilai atribut tersebut telah diubah setelah atribut tersebut dibaca dari database.
- Metode **`validate()`** akan tetap dilakukan pada semua atribut, tanpa melihat apakah atribut tersebut ***dirty*** atau tidak.
- Anda dapat mengetahui atribut mana saja yang ***dirty*** dengan memanggil metode **`yii\db\ActiveRecord::getDirtyAttributes()`**

Default Attribute Values

- Beberapa kolom pada table dapat memiliki nilai default.
- Untuk mendapatkan nilai default dari database dapat anda lakukan dengan memanggil metode **`loadDefaultValues()`**

```
$customer = new Customer();
$customer->loadDefaultValues();

// $customer->xyz will be assigned with default value declared for
// column "xyz" in the table “customer”
```

Meng-update Beberapa Baris Sekaligus

- Metode **`save()`** yang telah dijelaskan sebelumnya hanya dapat meng-update satu baris data untuk setiap pemanggilan.
- Untuk meng-update beberapa baris data sekaligus, dapat anda lakukan dengan memanggil metode **`updateAll()`**

```
// UPDATE `customer` SET `status` = 1 WHERE `email` LIKE`%@example.com%`
Customer::updateAll(
  ['status' => Customer::STATUS_ACTIVE],
  ['like', 'email','@example.com']
);
```

Menghapus Data

- Untuk menghapus satu baris data:
  - Ambil data yang akan dihapus dari database
  - Panggil metode **`yii\db\ActiveRecord::delete()`**
- Sedangkan untuk menghapus beberapa atau semua baris data sekaligus, anda dapat memanggil metode **`yii\db\ActiveRecord::deleteAll()`**

```
// Menghapus satu baris data pada tabel `customer` dengan `id` = 123
$customer = Customer::findOne(123);
$customer->delete();

// Menghapus semua baris data pada tabel `customer` yang kolom 'status'
// bernilai 0
Customer::deleteAll(['status' => Customer::STATUS_INACTIVE]);
```

Bekerja dengan Transaksi

- ***Transaction*** dapat diartikan sebagai: satu atau lebih perintah SQL yang harus selesai semuanya atau tidak sama sekali.
- Didalam sebuah ***transaction***, jika semua perintah berhasil dijalankan maka akan dilakukan ***commit***.
- Sedangkan jika ada satu saja perintah yang gagal, maka akan dilakukan perintah ***rollback***.
- Lihat contoh disamping untuk penggunaan transaksi pada AR.

```
$transaction = Customer::getDb()
->beginTransaction();
try {
  $customer->id = 200;
  $customer->save();
  // ...other DB operations...
  $transaction->commit();
} catch(\Exception $e) {
  $transaction->rollBack();
  throw $e;
}

// or alternatively
$customer = Customer::findOne(123);
Customer::getDb()->transaction(function($db) use
($customer) {
  $customer->id = 200;
  $customer->save();
  // ...other DB operations...
});
```

Bekerja dengan Relational Data

- Selain bekerja dengan table tunggal, Active Record juga mampu untuk membawa serta table yang berelasi dengan table utama.
- Sebagai contoh, data customer berelasi dengan data order. Dimana seseorang customer dapat memiliki satu atau lebih order.
- Dengan deklarasi relasi yang benar, anda dapat mengakses daftar order dari seorang customer melalui ekspresi `$customer->orders`. Ekspresi ini akan berisi array dari objek AR Order.

Deklarasi Relasi

- Untuk bekerja dengan relasi menggunakan AR, anda harus mendeklarasikan relasi di dalam class AR terlebih dulu.
- Langkah deklarasinya sederhana, cukup dengan mendeklarasikan metode relation untuk setiap table yang berelasi dengan table dari AR.
- Setiap metode untuk relasi harus diberi nama dengan format getXyz. Dimana "xyz" disebut sebagai nama relasi.
- Fungsi **`hasMany()`** digunakan untuk mengakses relasi ***One to Many***, sedangkan fungsi **`hasOne()`** digunakan untuk mengakses relasi ***One to One***.

```php
// class AR untuk table "customer"
class Customer extends ActiveRecord
{
  // ...
  public function getOrders()
  {
    return $this->hasMany(Order::className(),
      ['customer_id' => 'id']);
  }
}

// class AR untuk table "order"
class Order extends ActiveRecord
{
  // ...
  public function getCustomer()
  {
    return $this->hasOne(Customer::className(),
      ['id' => 'customer_id']);
  }
}
```

Mengakses Relational Data

- Setelah mendeklarasikan sebuah relasi, anda dapat mengakses relational data melalui nama relasinya.
- Jika sebuah relasi dideklarasikan menggunakan fungsi **`hasMany()`** maka dia akan mengembalikan array dari objek AR yang direlasikan.
  Sedangkan jika deklarasi relasi menggunakan fungsi **`hasOne()`** maka akan mengembalikan sebuah objek AR.
- Perhatikan contoh berikut ini:

```
// SELECT * FROM `customer` WHERE `id` = 123
$customer = Customer::findOne(123);

// SELECT * FROM `order` WHERE `customer_id` = 123
// $orders is an array of Order objects
$orders = $customer->orders;
```

Relasi melalui Junction Table

- Didalam Conceptual Data Modeling, kadang kita mendefinisikan relasi ***Many to Many***.
- Misal relasi antara table "order" dan table "item", dimana setiap order dapat berisi beberapa item dan satu item dapat berada dalam beberapa order.
- Pada kondisi ini, dibutuhkan sebuah table relasi yang biasa disebut sebagai ***Junction Table***.
- Anda dapat mendeklarasikan relation melalui Junction Table dengan memanggil metode **`via()`** atau **`viaTable()`**.

```
class Order extends ActiveRecord
{
  public function getItems()
  {
    return $this->hasMany(Item::className(), ['id'
      => 'item_id'])->viaTable('order_item',
        ['order_id' => 'id']);
  }
}


// or alternatifely

class Order extends ActiveRecord
{
  public function getOrderItems()
  {
    return $this->hasMany(OrderItem::className(),
      ['order_id' => 'id']);
  }

  public function getItems()
  {
    return $this->hasMany(Item::className(),
      ['id' => 'item_id'])->via('orderItems');
  }
}
```

Relasi melalui Junction Table (2)

- Cara penggunaan relasi yang dideklarasikan melalui junction table sama seperti relasi normal.
- Perhatikan contoh berikut ini:

```
// SELECT * FROM `order` WHERE `id` = 100
$order = Order::findOne(100);

// SELECT * FROM `order_item` WHERE `order_id` = 100
// SELECT * FROM `item` WHERE `item_id` IN (...)
// returns an array of Item objects
$items = $order->items;
```

Lazy Loading and Eager Loading

```php
// SELECT * FROM customer WHERE id = 123
$customer = Customer::findOne(123);

// SELECT * FROM order WHERE customer_id = 123
$orders = $customer->orders;

// no SQL executed
$orders2 = $customer->orders;
```

- Data pada table relasi dapat diakses seperti mengakses sebuah property normal.
- Pengaksesan seperti ini disebut sebagai ***Lazy Loading***. Yang artinya, query pada table relasi hanya akan dilakukan jika data pada table relasi tersebut dibutuhkan (dibaca) untuk pertama kali.
- Perhatikan contoh di samping atas:
  - Baris kode yang pertama hanya akan menjalankan SQL pada table "customer" saja.
  - Baris kode kedua akan menjalankan SQL pada table "order" dimana kolom customer_id = 123
  - Baris ketiga tidak menjalankan SQL apapun karena "$customer->orders" sudah pernah dilakukan sebelumnya.
- Penulisan kode seperti disamping disebut sebagai ***Lazy Loading***.
- ***Lazy Loading*** sangat convenient digunakan. Namun, untuk beberapa kasus, penggunaan ***Lazy Loading*** memiliki performa yang buruk.

Lazy Loading and Eager Loading (2)

```php
// SELECT * FROM customer LIMIT 100
$customers = Customer::find()->limit(100)->all();

foreach ($customers as $customer) {
  // SELECT * FROM order WHERE customer_id = ...
  $orders = $customer->orders;
}
```

- Perhatikan contoh di kanan atas. (yang dijalankan secara ***Lazy Loading***)
- Jika jumlah customer ada 100 orang, maka kode program tersebut akan menjalankan statemen SQL sebanyak 101 kali. → Tidak efisien.
- Hal ini disebabkan karena, setiap kali kita mengakses data order untuk masing-masing customer di dalam foreach, sebuah statemen SQL akan dijalankan.
- Masalah performa ini dapat diselesaikan dengan pendekatan ***Eager Loading***.
- Perhatikan contoh di kanan bawah.
- Dengan memanggil metode ***with()***, anda telah menginstruksikan objek AR untuk sekaligus mengakses data "order" dari 100 customer dengan satu statemen SQL.
- Sehingga total statemen SQL yang dijalankan hanya 2 saja, jauh lebih efisien dibandingkan dengan 101 statemen seperti yang dijalankan secara ***Lazy Loading***.

```php
// SELECT * FROM customer LIMIT 100;
// SELECT * FROM orders WHERE customer_id IN (...)
$customers = Customer::find()
  ->with('orders')
  ->limit(100)->all();

foreach ($customers as $customer) {
  // No SQL executed
  $orders = $customer->orders;
}
```

Latihan

- Buatlah tiga table:
  - customer (id INT, nama VARCHAR, email VARCHAR, user_id INT)
  - order (id INT, date DATETIME, customer_id INT)
  - order_item (order_id INT, item_id INT)
- Tentukan relasi antara table-table tersebut serta antara table customer dengan table user.
- Buat CRUD untuk customer, order, dan item.
  - Pastikan bahwa satu user hanya boleh membuat satu customer saja.
  - Ketika seorang user membuat data customer, id dari user tersebut akan ditambahkan pada kolom user_id di table customer secara otomatis.
- Tambahkan satu route (misal: customer/show-order) yang akan menampilkan daftar nama customer beserta nomor order dan daftar item yang dibelinya.

Terima Kasih