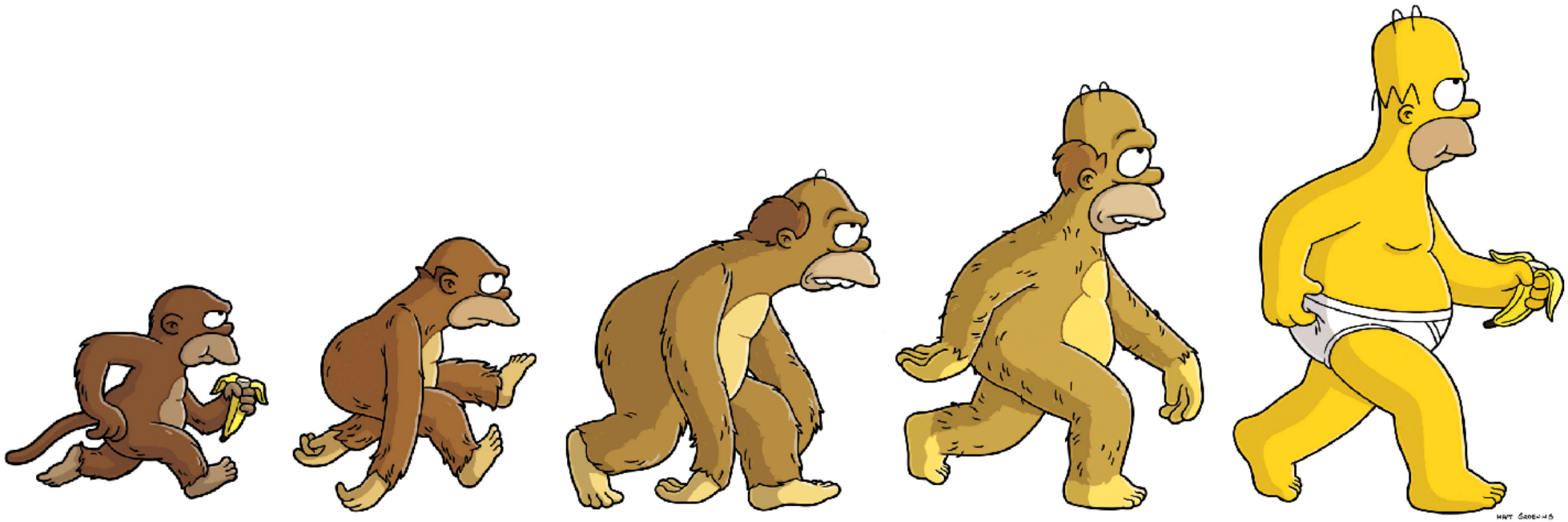
Apprendre en persistant



Niveau 1 Niveau 2 Niveau 3 : Ne pas Niveau 4 Niveau 5

Niveau 1
Parler le PDO

Fa

```php
<?php

$dsn = 'mysql:host=localhost;dbname=bookstore';
$user = 'fzaninotto';
$pass = 'S3Cr3t';

try {

    // opening the connection
    $con = new PDO($dsn, $user, $pass);
    $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $con->beginTransaction();

    // inserting an author
    $sql = 'INSERT INTO `author` (`first_name`, `last_name`)
      VALUES (:first_name, :last_name)';
    $sth = $con->prepare($sql);
    $sth->bindValue(':first_name', 'Leo', PDO::PARAM_STR);
    $sth->bindValue(':last_name', 'Tolstoi', PDO::PARAM_STR);
    $sth->execute();
```

```php
$user = 'root';
$pass = 'S3Cr3t';

try {

    // opening the connection
    $con = new PDO($dsn, $user, $pass);
    $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $con->beginTransaction();

    // inserting an author
    $sql = 'INSERT INTO `author` (`first_name`, `last_name`)
      VALUES (:first_name, :last_name)';
    $sth = $con->prepare($sql);
    $sth->bindValue(':first_name', 'Leo', PDO::PARAM_STR);
    $sth->bindValue(':last_name', 'Tolstoi', PDO::PARAM_STR);
    $sth->execute();

    // inserting a couple books
    $author_id = $con->lastInsertId() ;
    $sql = 'INSERT INTO `book` (`title`, `author_id`)
      VALUES (:title, :author_id)';
    $sth = $con->prepare($sql);
    $sth->bindValue(':title', 'War and Peace', PDO::PARAM_STR);
    $sth->bindValue(':author_id', $author_id, PDO::PARAM_INT);
    $sth->execute();
    $sth->bindValue(':title', 'Anna Karenina', PDO::PARAM_STR);
    $sth->bindValue(':author_id', $author_id, PDO::PARAM_INT);
```

```php
// inserting an author
$sql = 'INSERT INTO `author` (`first_name`, `last_name`)
  VALUES (:first_name, :last_name)';
$sth = $con->prepare($sql);
$sth->bindValue(':first_name', 'Leo', PDO::PARAM_STR);
$sth->bindValue(':last_name', 'Tolstoi', PDO::PARAM_STR);
$sth->execute();

// inserting a couple books
$author_id = $con->lastInsertId() ;
$sql = 'INSERT INTO `book` (`title`, `author_id`)
  VALUES (:title, :author_id)';
$sth = $con->prepare($sql);
$sth->bindValue(':title', 'War and Peace', PDO::PARAM_STR);
$sth->bindValue(':author_id', $author_id, PDO::PARAM_INT);
$sth->execute();
$sth->bindValue(':title', 'Anna Karenina', PDO::PARAM_STR);
$sth->bindValue(':author_id', $author_id, PDO::PARAM_INT);
$sth->execute();

// selecting a book by Tolstoi named 'War%'
$sql = 'SELECT `book`.`id`, `book`.`title`, `book`.`author_id`
  FROM `book`
  INNER JOIN `author` ON (`book`.`author_id` = `author`.`id`)
  WHERE `author`.`last_name` = ?
  AND `book`.`title` LIKE ?
```

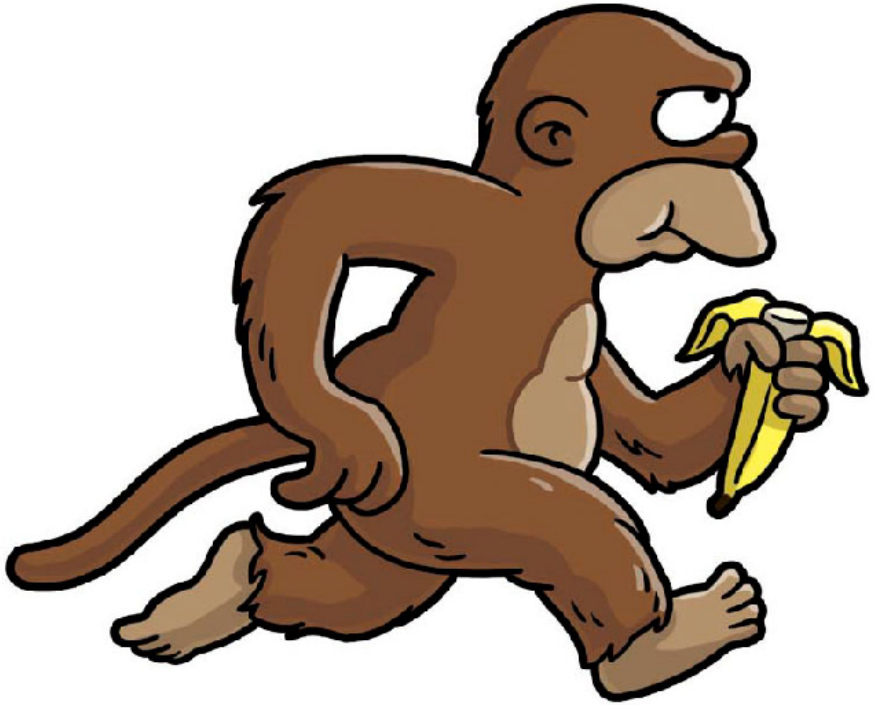
```
$sth->bindValue(':author_id', $author_id, PDO::PARAM_INT);
$sth->execute();
$sth->bindValue(':title', 'Anna Karenina', PDO::PARAM_STR);
$sth->bindValue(':author_id', $author_id, PDO::PARAM_INT);
$sth->execute();

// selecting a book by Tolstoi named 'War%'
$sql = 'SELECT `book`.`id`, `book`.`title`, `book`.`author_id`
  FROM `book`
  INNER JOIN `author` ON (`book`.`author_id` = `author`.`id`)
  WHERE `author`.`last_name` = ?
  AND `book`.`title` LIKE ?
  LIMIT 1';
$sth = $con->prepare($sql);
$sth->bindValue(1, 'Tolstoi', PDO::PARAM_STR);
$sth->bindValue(2, 'War%', PDO::PARAM_STR);
$sth->execute();
if ($row = $sth->fetch(PDO::FETCH_ASSOC)) {
  printf('Book of id %d, named %s, written by author of id %d',
    $row['id'], $row['title'], $row['author_id']
  );
}

// selecting all books
$sql = 'SELECT `book`.`id`, `book`.`title`, `book`.`author_id`
  FROM `book`';
$sth = $con->prepare($sql);
```

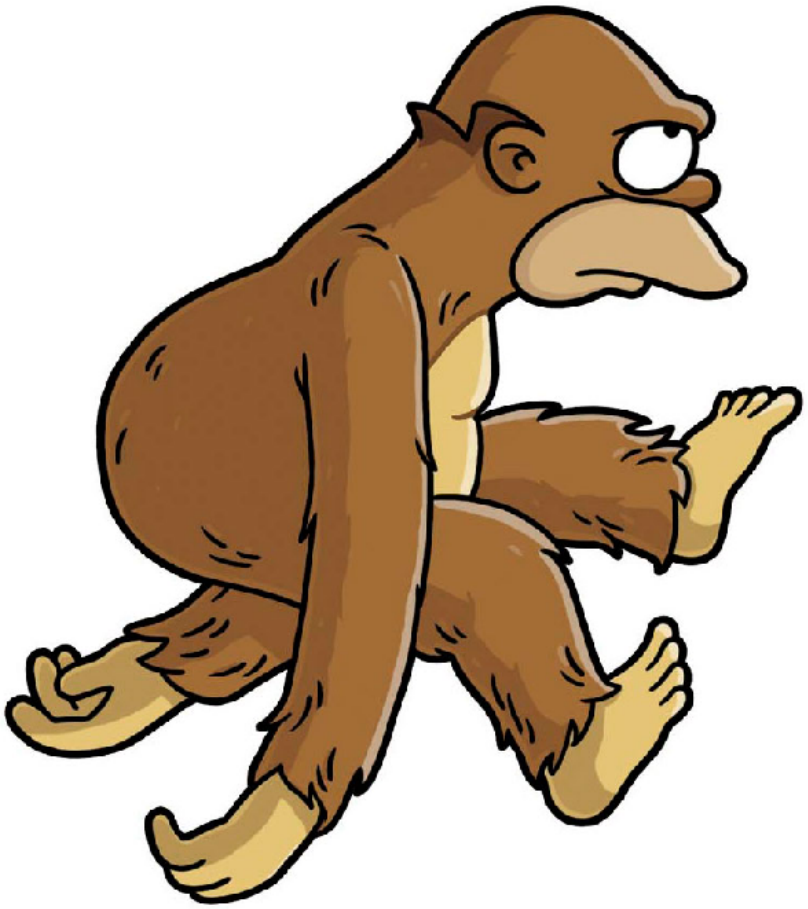
```php
$sth->execute();
if ($row = $sth->fetch(PDO::FETCH_ASSOC)) {
  printf('Book of id %d, named %s, written by author of id %d',
    $row['id'], $row['title'], $row['author_id']
  );
}

// selecting all books
$sql = 'SELECT `book`.`id`, `book`.`title`, `book`.`author_id`
  FROM `book`';
$sth = $con->prepare($sql);
$sth->execute();
$nbResults = 0;
while ($row = $sth->fetch(PDO::FETCH_NUM)) {
  $nbResults++;
  printf("Book of id %d, named %s, written by author of id %d\n",
    $row[0], $row[1], $row[2]
  );
}
if (!$nbResults) {
  echo "no result\n";
}

// commit the transaction
$con->commit();

// close the connection
$con = null;
```

```
}
if (!$nbResults) {
    echo "no result\n";
}

// commit the transaction
$con->commit();

// close the connection
$con = null;

} catch (PDOException $e) {

    $con->rollBack();
    printf("Failed: %s\n", $e->getMessage());
}
```

Niveau 1
Parler le PDO

Fa





Niveau 1
Parler le PDO

Niveau 2
Faire des objets

```php
<?php

/**
 * ActiveRecord class for the author table
 */
class Author
{
    public $id, $first_name, $last_name;

    /**
     * Get the Author's name
     *
     * @return string
     */
    public function getName()
```

```php
 * Activeke
 */
class Author
{
    public $id, $first_name, $last_name;

    /**
     * Get the Author's name
     *
     * @return string
     */
    public function getName()
    {
        if ($this->first_name && $this->last_name) {
            return $this->first_name . ' ' . $this->last_name;
        } elseif ($this->last_name) {
            return $this->last_name;
        } elseif ($this->first_name) {
            return $this->first_name;
        } else {
            return 'Mr. Nobody';
        }
    }

    /**
     * from a resultset into the current object
     * fetched using PDO::FETCH_NUM
```

```
    }
}

/**
 * Load data from a resultset into the current object
 * The resultset should be fetched using PDO::FETCH_NUM
 *
 * @param array $row numerically indexed array of values
 */
public function load($row)
{
    $this->id         = $row[0];
    $this->first_name = $row[1];
    $this->last_name  = $row[2];
}

/**
 * Persist the data of the current object to a database
 *
 * @param  PDO $con Connection to use
 */
public function save(PDO $con)
{
    if (null === $this->id) {
        return $this->insert($con);
    } else {
        return $this->update($con);
    }
}

of the current object as a new record
```

```php
 */
public function save(PDO $con)
{
    if (null === $this->id) {
        return $this->insert($con);
    } else {
        return $this->update($con);
    }
}

/**
 * Insert the data of the current object as a new record
 *
 * @param  PDO $con Connection to use
 */
public function insert(PDO $con)
{
    $sql = 'INSERT INTO `author` (`first_name`, `last_name`)
        VALUES (:first_name, :last_name)';
    $stmt = $con->prepare($sql);
    $this->bindValue($stmt, ':first_name', $this->first_name, PDO::PARAM_STR);
    $this->bindValue($stmt, ':last_name', $this->last_name, PDO::PARAM_STR);
    $stmt->execute();
    $this->id = $con->lastInsertId();
}

/**
 * Update an existing record with the data from this object
 *
 * @param  PDO $con Connection to use
 */
public function update(PDO $con)
{
    if (null === $this->id) {
        throw new PropelException('Cannot update an object without a Pk');
```

```
        VALUES (
        $stmt = $con->prepare($
        $this->bindValue($stmt, ':first_name', $this->last_na
        $this->bindValue($stmt, ':last_name', $this->last_name);
        $stmt->execute();
        $this->id = $con->lastInsertId();
    }

    /**
     * Update an existing record with the data from this object
     *
     * @param  PDO $con Connection to use
     */
    public function update(PDO $con)
    {
        if (null === $this->id) {
            throw new Exception('Cannot update an object without a Pk');
        }
        $sql = 'UPDATE `author`
            SET `first_name` = :first_name, `last_name` = :last_name
            WHERE `id` = :id';
        $stmt = $con->prepare($sql);
        $stmt->bindValue(':id', $this->id, PDO::PARAM_INT);
        $this->bindValue($stmt, ':first_name', $this->first_name, PDO::PARAM_STR);
        $this->bindValue($stmt, ':last_name', $this->last_name, PDO::PARAM_);
        $stmt->execute();
    }

    /**
     * Binds a value to a parameter, handling NULL values the right way
     *
     * @param PDOStatement $stmt The statement to bind on
     * @param integer|string $parameter Parameter indentifier
     * @param mixed $value The value to bind to the parameter
     * @param integer $dataType Explicit data type for the parameter using the
     */
    protected function bindValue($stmt, $parameter, $value = null,
```

```php
public function
{
    if (null === $this->id) {
        throw new Exception('Cannot update an object without
    }
    $sql = 'UPDATE `author`
        SET `first_name` = :first_name, `last_name` = :last_name
        WHERE `id` = :id';
    $stmt = $con->prepare($sql);
    $stmt->bindValue(':id', $this->id, PDO::PARAM_INT);
    $this->bindValue($stmt, ':first_name', $this->first_name, PDO::PARAM_STR);
    $this->bindValue($stmt, ':last_name', $this->last_name, PDO::PARAM_);
    $stmt->execute();
}

/**
 * Binds a value to a parameter, handling NULL values the right way
 *
 * @param PDOStatement $stmt The statement to bind on
 * @param integer|string $parameter Parameter indentifier
 * @param mixed $value The value to bind to the parameter
 * @param integer $dataType Explicit data type for the parameter using the PDO::PARAM_* constants.
 */
protected function bindValue(PDOStatement $stmt, $parameter, $value = null, $dataType = PDO::PARAM_STR)
{
    $dataType = (null === $value) ? PDO::PARAM_NULL : $dataType;
    $stmt->bindValue($parameter, $value, $dataType);
}
}
```

class for the book table

$title, $author_id;

a from a resultset into the current object
d be fetched using PDO::FETCH_NUM

```php
<?php

/**
 * ActiveRecord class for the book table
 */
class Book
{
    public $id, $title, $author_id;

    /**
     * Load data from a resultset into the current object
     * The resultset should be fetched using PDO::FETCH_NUM
     *
     * @param array $row numerically indexed array of values
     */
    public function load($row)
    {
        $this->id = $row[0];
        $this->title = $row[1];
        $this->author_id = $row[2];
    }

    /**
     * Persist the data of the current object to a database
     *
     * @param  PDO $con Connection to use
     */
    public function save(PDO $con)
    {
        if ($this->id) {
```

```php
/**
 * Insert the data of the
 *
 * @param  PDO $con Connection to use
 */
public function insert(PDO $con)
{
    $sql = 'INSERT INTO `book` (`title`, `author_id`)
        VALUES (:title, :author_id)';
    $stmt = $con->prepare($sql);
    $this->bindValue($stmt, ':title', $this->title, PDO::PARAM_STR);
    $this->bindValue($stmt, ':author_id', $this->author_id, PDO::PARAM_INT);
    $stmt->execute();
    $this->id = $con->lastInsertId();
}

/**
 * Update an existing record with the data from this object
 *
 * @param  PDO $con Connection to use
 */
public function update(PDO $con)
{
    if (null === $this->id) {
        throw new Exception('Cannot update an object without a Pk');
    }
    $sql = 'UPDATE `book`
        SET `title` = :title, `author_id` = :author_id
        WHERE `id` = :id';
    $stmt = $con->prepare($sql);
    $stmt->bindValue(':id', $this->id, PDO::PARAM_INT);
    $this->bindValue($stmt, ':title', $this->title, PDO::PARAM_STR);
    $this->bindValue($stmt, ':author_id', $this->author_id, PDO::PARAM_INT);
    $stmt->execute();
}

/**
 * value to a parameter, handling NULL values the right way
 *
 * ment to bind on
 * ifier
```

```
        $stmt->bindValue($param
    }

    /**
     * Get the related Author object if exists. Uses Lazy-loading.
     *
     * @param  PDO $con Connection to use
     * @return null|Author
     * @throws Exception If the author id doesn't exist in the database
     */
    public function getAuthor(PDO $con)
    {
        if (null === $this->author_id) {
            return null;
        }
        $sql = 'SELECT author.id, author.first_name, author.last_name
            FROM author
            WHERE author.id = :id
            LIMIT 1';
        $stmt = $con->prepare($sql);
        $stmt->bindValue(':id', $this->author_id, PDO::PARAM_INT);
        $stmt->execute();
        if ($row = $stmt->fetch(PDO::FETCH_NUM)) {
            require_once 'Author.php';
            $author = new Author();
            $author->load($row);
            return $author;
        } else {
            throw new Exception(sprintf(
                'No author for this book\'s author_id, %d',
                $this->author_id
            ));
        }
    }
}
```

```php
<?php

/**
 * QueryObject for ActiveRecord queries
 */
class Query
{
  protected
    $resultClass,
    $from,
    $wheres = array(),
    $joins = array(),
    $limit;

  /**
   * Query Constructor
   * Requires an ActiveRecord class name. Beware that the cl
   *
   * @param string $resultClass An ActiveRecord class, e.g.
```

```
{
  protected
    $resultClass,
    $from,
    $wheres = array(),
    $joins = array(),
    $limit;

  /**
   * Query Constructor
   * Requires an ActiveRecord class name. Beware that the class file is not included.
   *
   * @param string $resultClass An ActiveRecord class, e.g. 'Book'
   * @param string $tableName The name of the table to query for.
   *               Defaults to the lowercased $resultClass
   */
  public function __construct($resultClass, $tableName = null)
  {
    $this->resultClass = $resultClass;
    if (null === $tableName) {
      $tableName = strtolower($resultClass);
    }
    $this->from = $tableName;
  }

  /**
   * Adds a WHERE clause.
   * <code>$query->where('book.title LIKE ?', 'foo%', PDO::PARAM_STR)</code>
   *
   * @param string $clause The SQL clause, using question mark as placeholder.
   * @param scalar $value The value to bind. Does not accept arrays
   * @param integer $type The PDO type used for binding
   */
  public function where($clause, $value = null, $type = PDO::PARAM_STR)
  {
```

```
     */
    public function __construct($resultClass, $tableName = null)
    {
      $this->resultClass = $resultClass;
      if (null === $tableName) {
        $tableName = strtolower($resultClass);
      }
      $this->from = $tableName;
    }

    /**
     * Adds a WHERE clause.
     * <code>$query->where('book.title LIKE ?', 'foo%', PDO::PARAM_STR)</code>
     *
     * @param string $clause The SQL clause, using question mark as placeholder.
     * @param scalar $value The value to bind. Does not accept arrays
     * @param integer $type The PDO type used for binding
     */
    public function where($clause, $value = null, $type = PDO::PARAM_STR)
    {
      $this->wheres[] = array(
        'clause' => $clause,
        'value'  => $value,
        'type'   => $type
      );
    }

    /**
     * Adds a limit to the query
     *
     * @param integer $limit The numerical limit
     */
    public function limit($limit)
    {
      $this->limit = (int) $limit;
    }
```

```
      'value'   => $value,
      'type'    => $type
    );
  }

  /**
   * Adds a limit to the query
   *
   * @param integer $limit The numerical limit
   */
  public function limit($limit)
  {
    $this->limit = (int) $limit;
  }

  public function join($foreignTable, $clause, $type = 'INNER')
  {
    $this->joins[]= array(
      'foreignTable' => $foreignTable,
      'clause'       => $clause,
      'type'         => $type
    );
  }

  /**
   * Execute the query and returns a list of ActiveRecord objects
   * based on the result.
   *
   * @param PDO $con Connection to use
```

```
      'type'          => $type
    );
  }

  /**
   * Execute the query and returns a list of ActiveRecord objects
   * based on the result.
   *
   * @param PDO $con Connection to use
   * @return array List of ActiveRecord objects
   */
  public function find(PDO $con)
  {
    $class = $this->resultClass;
    $stmt = $this->doExecute($con);
    $result = array();
    while ($row = $stmt->fetch(PDO::FETCH_NUM)) {
      $record  = new $class();
      $record->load($row);
      $result[]= $record;
    }
    return $result;
  }

  /**
   * Execute the query and returns an ActiveRecord object
   * based on the result.
```

```
      $record->load($row);
      $result[]= $record;
    }
    return $result;
}

/**
 * Execute the query and returns an ActiveRecord object
 * based on the result.
 *
 * @param PDO $con Connection to use
 * @return mixed An ActiveRecord object, or null if no result
 */
public function findOne(PDO $con)
{
    $class = $this->resultClass;
    $stmt = $this->doExecute($con);
    if ($row = $stmt->fetch(PDO::FETCH_NUM)) {
      $record  = new $class();
      $record->load($row);
      return $record;
    }
}

/**
 * Prepares, binds and executes the SQL query
 *
 * @param PDO $con
```

```
      $record = new $class();
      $record->load($row);
      return $record;
    }
  }

  /**
   * Prepares, binds and executes the SQL query
   *
   * @param PDO $con
   * @return PDOStatement
   */
  protected function doExecute(PDO $con)
  {
    $stmt = $con->prepare($this->getSQL());
    foreach ($this->wheres as $key => $where) {
      if (null === $where['value']) {
        $stmt->bindValue($key + 1, null, PDO::PARAM_NULL);
      } else {
        $stmt->bindValue($key + 1, $where['value'], $where['type']);
      }
    }
    $stmt->execute();
    return $stmt;
  }

  /**
   * Creates the SQL string based on the current object's properties
   *
```

```php
  }

  /**
   * Creates the SQL string based on the current object's properties
   *
   * @return string the SQL string to execute
   */
  public function getSQL()
  {
    // build the SELECT FROM part
    $sql = sprintf('SELECT * FROM `%s`', $this->from);
    // build the JOIN part
    foreach ($this->joins as $join) {
      $sql .= sprintf(' %s JOIN `%s` ON (%s)',
        $join['type'],
        $join['foreignTable'],
        $join['clause']
      );
    }
    // build the WHERE part
    if ($this->wheres) {
      $whereClauses = array();
      foreach ($this->wheres as $key => $where) {
        $whereClauses []= $where['clause'];
      }
      $sql .= ' WHERE ' . implode(' AND ', $whereClauses);
    }
    // build the LIMIT part
    if ($this->limit) {
      $sql .= ' LIMIT ' . $this->limit;
    }
    return $sql;
  }
}
```

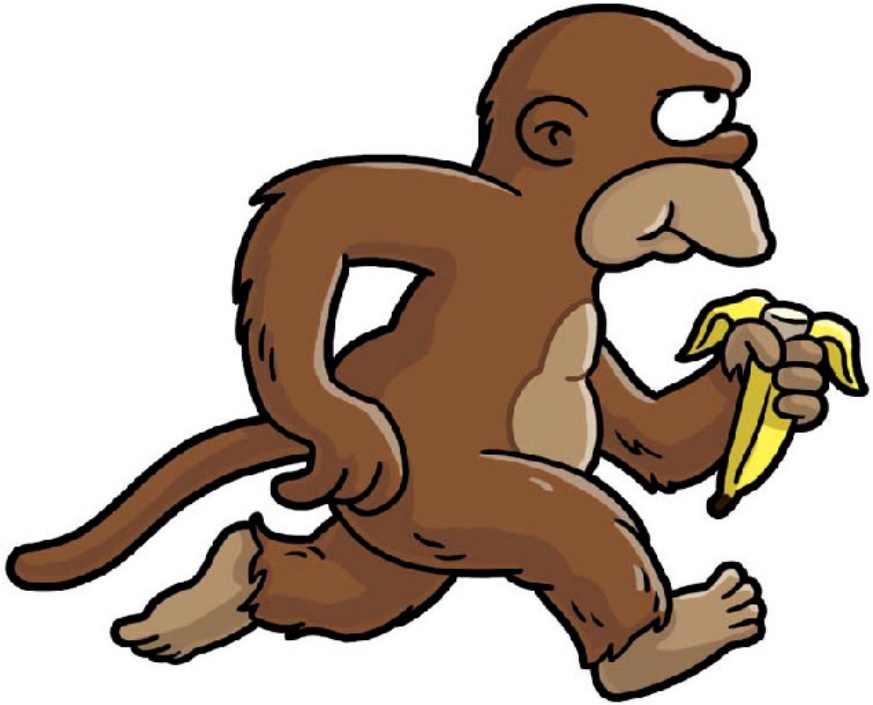
```php
<?php

require_once 'level2/Book.php';
require_once 'level2/Author.php';
require_once 'level2/Query.php';

$dsn = 'mysql:host=localhost;dbname=bookstore';
$user = 'fzaninotto';
$pass = 'S3Cr3t';

try {

    // opening the connection
    $con = new PDO($dsn, $user, $pass);
    $con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTIO
    $con->beginTransaction();

    // inserting an author
    $author = new Author();
    $author->first_name = 'Leo';
    $author->last_name = 'Tolstoi';
```

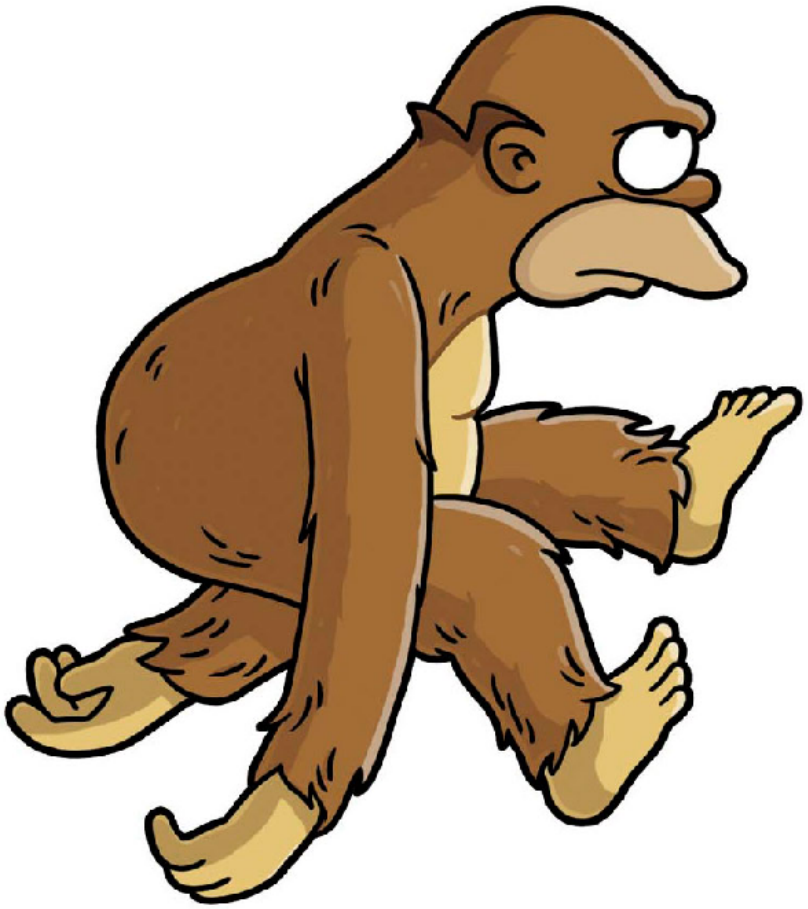
```php
try {
    // opening the connection
    $con = new PDO($dsn, $user, $pass);
    $con->setAttribute(PDO::ATTR_ERRMODE,
    $con->beginTransaction();

    // inserting an author
    $author = new Author();
    $author->first_name = 'Leo';
    $author->last_name = 'Tolstoi';
    $author->save($con);

    // inserting a couple books
    $book1 = new Book();
    $book1->title = 'War and Peace';
    $book1->author_id = $author->id;
    $book1->save($con);
```

```
$author->first_name = 'Tolstoi';
$author->last_name = 'Tolstoi';
$author->save($con);

// inserting a couple books
$book1 = new Book();
$book1->title = 'War and Peace';
$book1->author_id = $author->id;
$book1->save($con);
$book2 = new Book();
$book2->title = 'Anna Karenina';
$book2->author_id = $author->id;
$book2->save($con);

// selecting a book by Tolstoi name
$query = new Query('Book');
$query->join('author', '`book`.`aut
                        `last_name`
```

```php
$author

// inserting a couple books
$book1 = new Book();
$book1->title = 'War and Peace';
$book1->author_id = $author->id;
$book1->save($con);
$book2 = new Book();
$book2->title = 'Anna Karenina';
$book2->author_id = $author->id;
$book2->save($con);

// selecting a book by Tolstoi named 'War%'
$query = new Query('Book');
$query->join('author', '`book`.`author_id` = `author`.`id`');
$query->where('`author`.`last_name` = ?', 'Tolstoi', PDO::PARAM_STR);
$query->where('`book`.`title` LIKE ? ', 'War%', PDO::PARAM_STR);
if ($book = $query->findOne($con)) {
    printf('Book of id %d, named %s, written by %s',
        $book->id, $book->title, $book->getAuthor($con)->getName()
    );
}

// selecting all books
$query = new Query('Book');
$books = $query->find($con);
foreach ($books as $book) {
    printf("Book of id %d, named %s, written by %s\n",
        $book->id, $book->title, $book->getAuthor($con)->getName()
    );
```

```
$query->join('au
$query->where('`author`.`last_
$query->where('`book`.`title` LIKE ? ', 'wa
if ($book = $query->findOne($con)) {
    printf('Book of id %d, named %s, written by %s',
        $book->id, $book->title, $book->getAuthor($con)->getName()
    );
}

// selecting all books
$query = new Query('Book');
$books = $query->find($con);
foreach ($books as $book) {
    printf("Book of id %d, named %s, written by %s\n",
        $book->id, $book->title, $book->getAuthor($con)->getName()
    );
}
if (!$books) {
    echo "no result\n";
}

// commit the transaction
$con->commit();

// close the connection
$con = null;

catch (PDOException $e) {
```

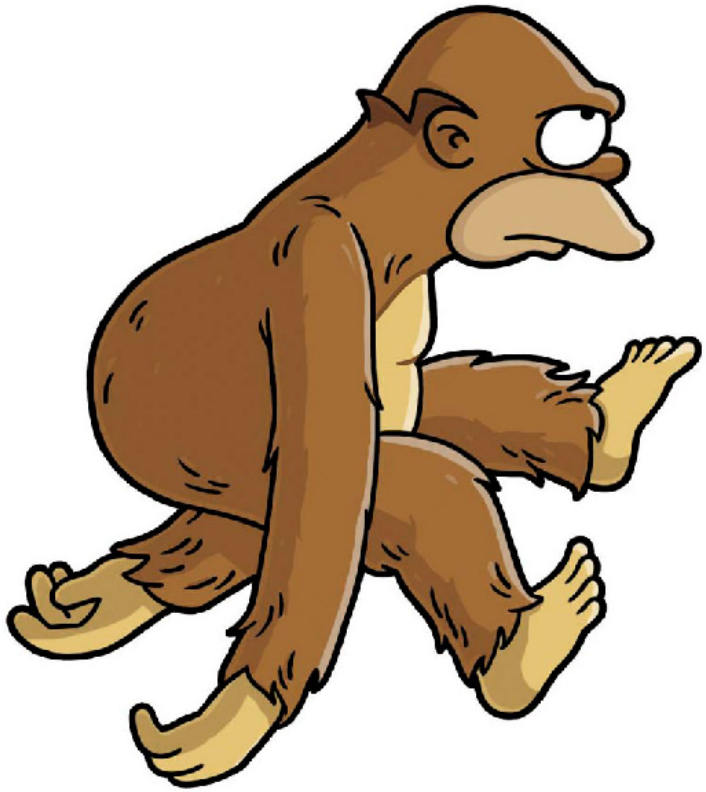




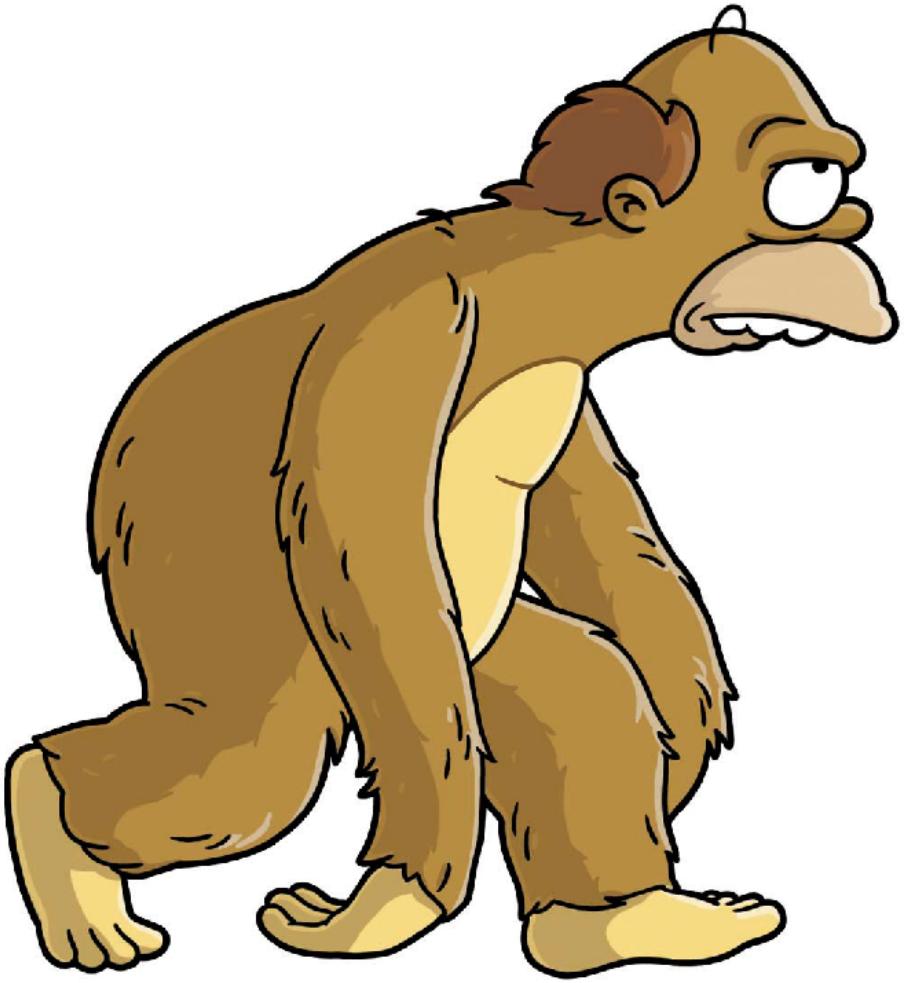
Niveau 1
Parler le PDO

Niveau 2
Faire des objets

1
PDO



Niveau 2
Faire des objets



Niveau 3 : Ne pas
réinventer la roue

- 7 ans d'âge
- Licence MIT
- Compatible PHP 5.2 et 5.3, MySQL, PostgreSQL, SQLite, Oracle, MSSQL
- Rapide comme l'éclair
- 125 000 lignes de code
- 3800 tests unitaires
- Documentation complète
- Utilisé par des milliers de sites web
- Utilisé par Symfony

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<database name="bookstore" defaultIdMethod="native"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.propelorm.org/xsd/1.5/database.xsd">

  <table name="book">
    <column name="id" required="true" primaryKey="true" autoIncrement="true" type="INTEGER"
    <column name="title" type="VARCHAR" required="true" primaryString="true" />
    <column name="author_id" required="false" type="INTEGER" />
    <foreign-key foreignTable="author" onDelete="CASCADE">
      <reference local="author_id" foreign="id" />
    </foreign-key>
  </table>

  <table name="author">
    <column name="id" required="true" primaryKey="true" autoIncrement="true" type="INTEGER" />
    <column name="first_name" required="true" type="VARCHAR" size="128" />
    <column name="last_name" required="true" type="VARCHAR" size="128" />
  </table>

</database>
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<config>
```

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<config>
  <propel>
    <datasources default="bookstore">
      <datasource id="bookstore">
        <adapter>mysql</adapter>
        <connection>
          <dsn>mysql:host=localhost;dbname=bookstore</dsn>
          <user>fzaninotto</user>
          <password>S3Cr3t</password>
        </connection>
      </datasource>
    </datasources>
  </propel>
</config>
```

```
$ ../vendor/propel/generator/bin/propel-gen om
```

```
</config>
```

```
$ ../vendor/propel/generator/bin/propel-gen om

propel > om:

     [echo] +-------------------------------------------+
     [echo] |                                           |
     [echo] | Generating Peer-based Object Model for    |
     [echo] | YOUR Propel project!                      |
     [echo] |                                           |
     [echo] +-------------------------------------------+
[phingcall] Calling Buildfile '/path/to/propel/generator/build-propel.xml' with target 'om-template'
 [property] Loading /path/to/propel/generator/./default.properties

propel > om-template:

[propel-om] Processing: schema.xml
[propel-om] Processing Datamodel : schema.xml
[propel-om]   - processing database : bookstore
[propel-om]     + book
[propel-om]             -> BaseBookPeer [builder: PHP5PeerBuilder]
[propel-om]             -> BaseBook [builder: PHP5ObjectBuilder]
[propel-om]             -> BookTableMap [builder: PHP5TableMapBuilder]
[propel-om]             -> BaseBookQuery [builder: QueryBuilder]
[propel-om]     + author
[propel-om]             -> BaseAuthorPeer [builder: PHP5PeerBuilder]
[propel-om]             -> BaseAuthor [builder: PHP5ObjectBuilder]
[propel-om]             -> AuthorTableMap [builder: PHP5TableMapBuilder]
[propel-om]             -> BaseAuthorQuery [builder: QueryBuilder]
[propel-om] Object model generation complete - 8 files written

BUILD FINISHED

Total time: 0.4737 seconds



$ ../vendor/propel/generator/bin/propel-gen convert-conf
```

```
BUILD FINISHED

Total time: 0.4737 seconds



$ ../vendor/propel/generator/bin/propel-gen convert-conf

propel > convert-conf:

     [echo] +-------------------------------------------+
     [echo] |                                           |
     [echo] | Converting runtime config file to an      |
     [echo] | array dump for improved performance.      |
     [echo] |                                           |
     [echo] +-------------------------------------------+
     [echo] Output file: bookstore-conf.php
     [echo] XMLFile: /path/to/level3/runtime-conf.xml
[propel-convert-conf] Creating PHP runtime conf file: /path/to/level3/build/conf/bookstore-conf.php
[propel-convert-conf] Processing: schema.xml
[propel-convert-conf] Adding class mapping: BookTableMap => bookstore/map/BookTableMap.php
[propel-convert-conf] Adding class mapping: BookPeer => bookstore/BookPeer.php
[propel-convert-conf] Adding class mapping: Book => bookstore/Book.php
[propel-convert-conf] Adding class mapping: BookQuery => bookstore/BookQuery.php
[propel-convert-conf] Adding class mapping: BaseBookPeer => bookstore/om/BaseBookPeer.php
[propel-convert-conf] Adding class mapping: BaseBook => bookstore/om/BaseBook.php
[propel-convert-conf] Adding class mapping: BaseBookQuery => bookstore/om/BaseBookQuery.php
[propel-convert-conf] Adding class mapping: AuthorTableMap => bookstore/map/AuthorTableMap.php
[propel-convert-conf] Adding class mapping: AuthorPeer => bookstore/AuthorPeer.php
[propel-convert-conf] Adding class mapping: Author => bookstore/Author.php
[propel-convert-conf] Adding class mapping: AuthorQuery => bookstore/AuthorQuery.php
[propel-convert-conf] Adding class mapping: BaseAuthorPeer => bookstore/om/BaseAuthorPeer.php
[propel-convert-conf] Adding class mapping: BaseAuthor => bookstore/om/BaseAuthor.php
[propel-convert-conf] Adding class mapping: BaseAuthorQuery => bookstore/om/BaseAuthorQuery.php
[propel-convert-conf] Creating PHP classmap runtime file: /path/to/level3/build/conf/classmap-bookstore-conf.php

BUILD FINISHED

Total time: 0.1997 seconds

$
```

```
BUILD FINISHED

Total time: 0.1997 seconds

$
```

```php
<?php

require_once 'vendor/propel/runtime/lib/Propel.php';
set_include_path(dirname(__FILE__) . '/level3/build/classes' . PATH_SEPARATOR .
Propel::init(dirname(__FILE__) . '/level3/build/conf/bookstore-conf.php');

try {

  // opening the connection
  $con = Propel::getConnection();
  $con->beginTransaction();

  // inserting an author
  $author = new Author();
  $author->setFirstName('Leo');
  $author->setLastName('Tolstoi');
  $author->save($con);

  // inserting a couple books
  $book1 = new Book();
  $book1->setTitle('War and Peace');
```

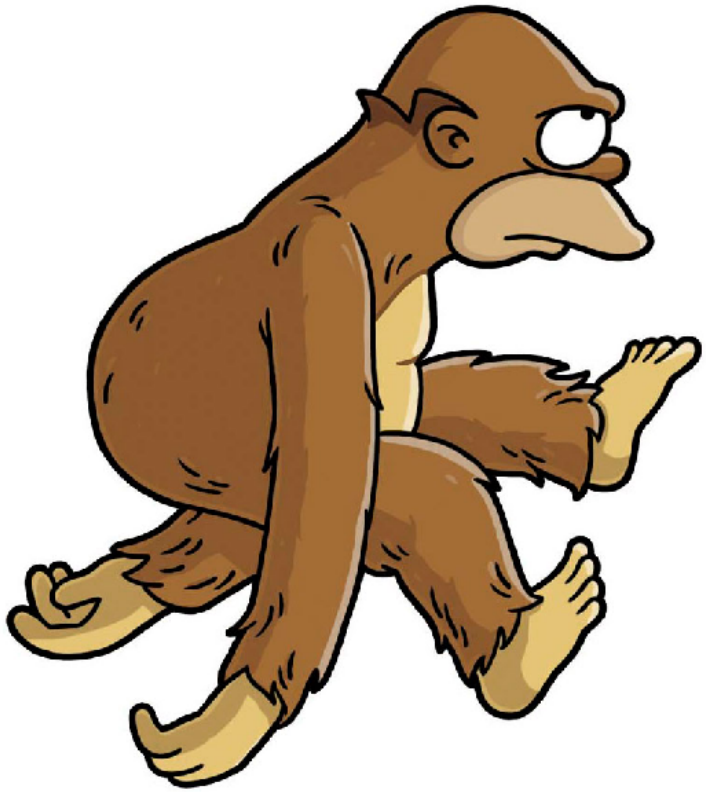
```php
try {

    // opening the connection
    $con = Propel::getConnection();
    $con->beginTransaction();

    // inserting an author
    $author = new Author();
    $author->setFirstName('Leo');
    $author->setLastName('Tolstoi');
    $author->save($con);

    // inserting a couple books
    $book1 = new Book();
    $book1->setTitle('War and Peace');
    $book1->setAuthor($author);
    $book1->save($con);
```

```php
$author = new Author();
$author->setFirstName('Leo');
$author->setLastName('Tolstoi');
$author->save($con);

// inserting a couple books
$book1 = new Book();
$book1->setTitle('War and Peace');
$book1->setAuthor($author);
$book1->save($con);
$book2 = new Book();
$book2->setTitle('Anna Karenina');
$book2->setAuthor($author);
$book2->save($con);

// selecting a book by Tolstoi named 'W
$book = BookQuery::create()
  ->useAuthorQuery()
    ->filterByLastName('Toslt oi')
```

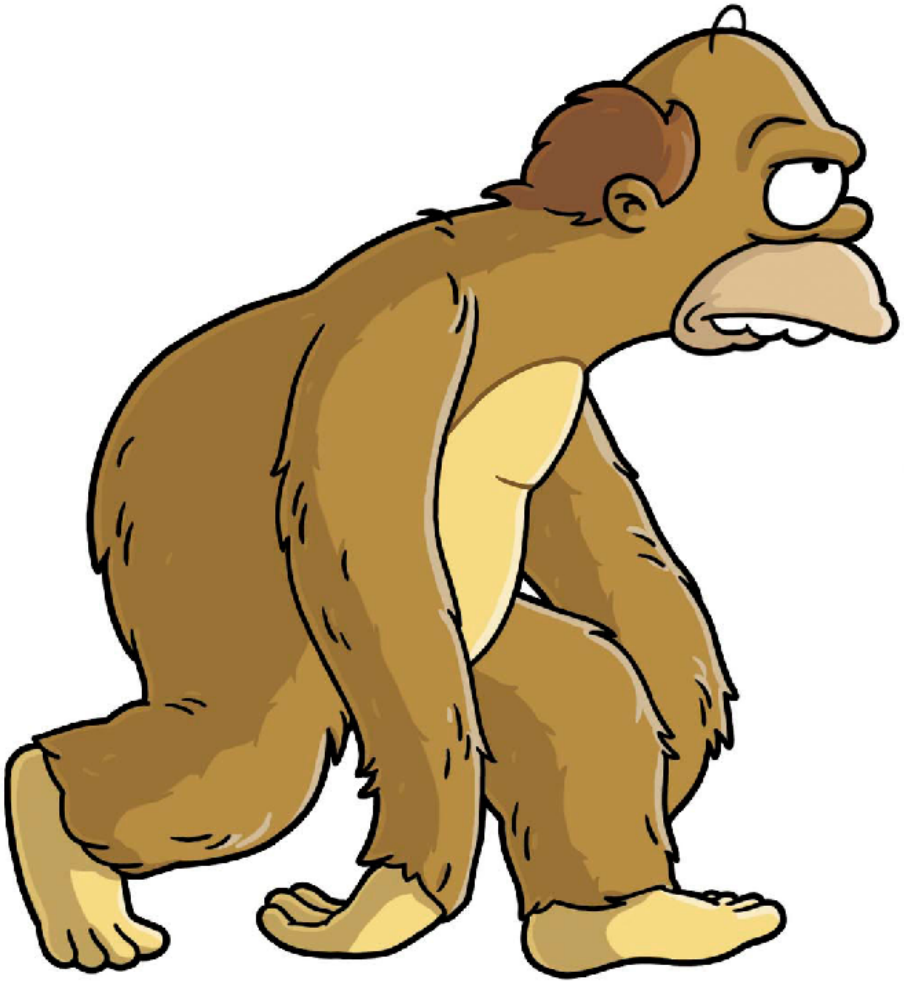
```php
$book1 = new Book();
$book1->setTitle('War and Peace');
$book1->setAuthor($author);
$book1->save($con);
$book2 = new Book();
$book2->setTitle('Anna Karenina');
$book2->setAuthor($author);
$book2->save($con);

// selecting a book by Tolstoi named 'War%'
$book = BookQuery::create()
  ->useAuthorQuery()
    ->filterByLastName('Toslltoi')
  ->endUse()
  ->filterByTitle('War%')
  ->with('Author')
  ->findOne($con);
if ($book) {
  printf('Book of id %d, named %s, written by %s',
    $book->getId(), $book->getTitle(), $book->getAuthor($con)->getName()
  );
}

// selecting all books
$books = BookQuery::create()->find($con);
foreach ($books as $book) {
  printf("Book of id %d, named %s, written by %s\n",
    $book->getId(), $book->getTitle(), $book->getAuthor($con)->getName()
  );
}
```

```
    ->endUse()
    ->filterByTitle('War%')
    ->with('Author')
    ->findOne($con);
  if ($book) {
    printf('Book of id %d, named %s, written by %s',
      $book->getId(), $book->getTitle(), $book->getAuthor($con)->getName()
    );
  }

  // selecting all books
  $books = BookQuery::create()->find($con);
  foreach ($books as $book) {
    printf("Book of id %d, named %s, written by %s\n",
      $book->getId(), $book->getTitle(), $book->getAuthor($con)->getName()
    );
  }
  if (!$books->count()) {
    echo "no result\n";
  }

  // commit the transaction
  $con->commit();

} catch (PropelException $e) {

  $con->rollBack();
  printf("Failed: %s\n", $e->getMessage());
}
```

1
PDO

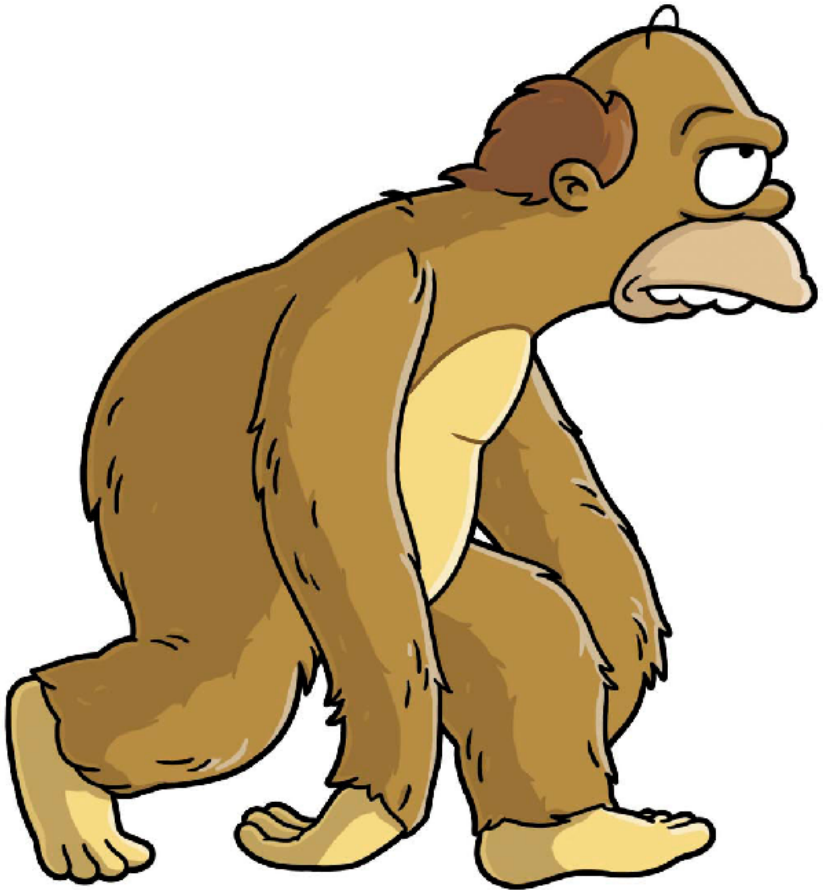


Niveau 2
Faire des objets

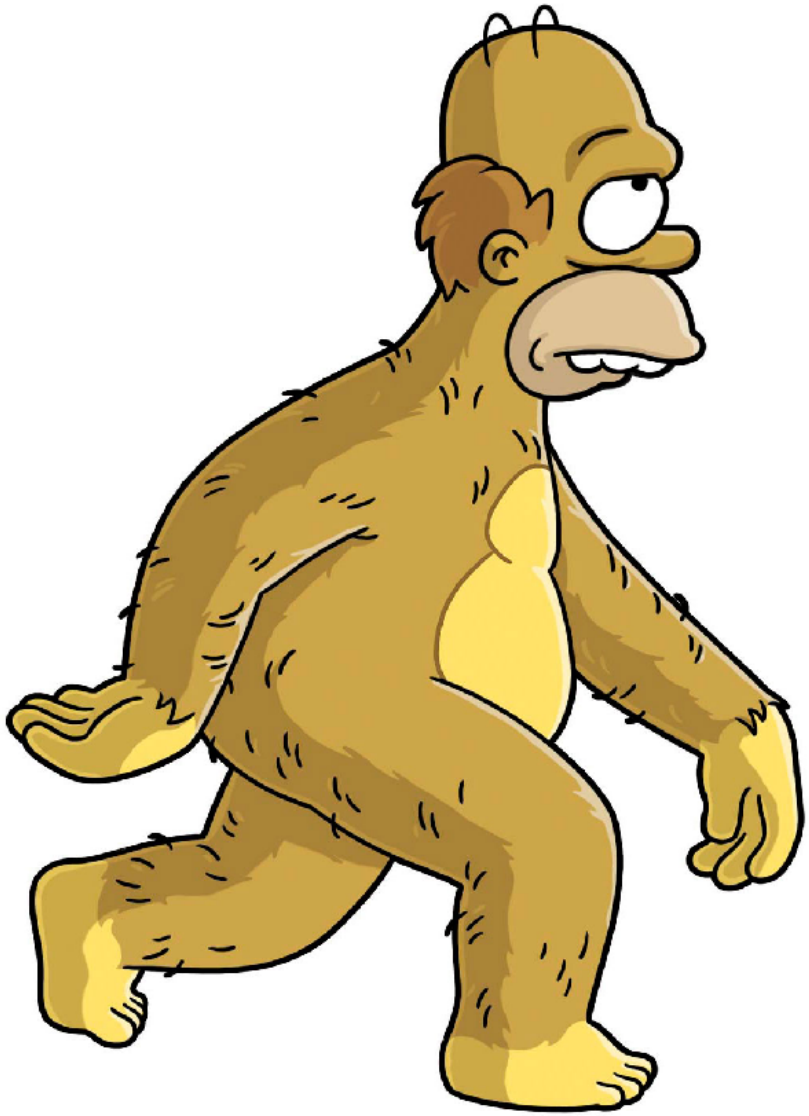


Niveau 3 : Ne pas
réinventer la roue

2
bjets



Niveau 3 : Ne pas
réinventer la roue



Niveau 4
Maitriser l'outil

```php
<?php

require_once 'vendor/propel/runtime/lib/Propel.php';
set_include_path(dirname(__FILE__) . '/level4/build/clas
Propel::init(dirname(__FILE__) . '/level4/build/conf/boo

// namespaces
use Bookstore\Book;
use Bookstore\BookQuery;
use Bookstore\Author;
use Bookstore\AuthorQuery;

// implicit connection

// Cascade save
$book1 = new Book();
$book1->setTitle('Harry Potter and the Philosopher\'s
$book2 = new Book();
$book2->setTitle('Harry Potter and the Chamber of Sec
```

```php
use Bookstore\BookQuery;
use Bookstore\Author;
use Bookstore\AuthorQuery;

// implicit connection

// Cascade save
$book1 = new Book();
$book1->setTitle('Harry Potter and the Philosopher\'s Stone');
$book2 = new Book();
$book2->setTitle('Harry Potter and the Chamber of Secrets');
$author = new Author();
$author->setFirstName('J.K.');
$author->setLastName('Rowling');
$author->addBook($book1);
$author->addBook($book2);
$author->save();

// Collections
$books = $author->getBooks();
echo $books->isEmpty(); // false
echo $books->count(); // 2
echo $books->getFirst()->getTitle(); // Harry Potter and the P
foreach ($books as $book) {
  printf('<li class="%s">%s</li>',
```

```php
$author->setLastName('Rowling');
$author->addBook($book1);
$author->addBook($book2);
$author->save();

// Collections
$books = $author->getBooks();
echo $books->isEmpty(); // false
echo $books->count(); // 2
echo $books->getFirst()->getTitle(); // Harry Potter and the Phil
foreach ($books as $book) {
    printf('<li class="%s">%s</li>',
        $books->isOdd() ? 'odd' : 'even',
        $book->getTitle()
    );
}
//<li class="even">Harry Potter and the Philosopher's Stone</li>
//<li class="odd">Harry Potter and the Chamber of Secrets</li>
unset($books[1]);
unset($books[0]);
$books []= $book1;

// Query termination methods
echo BookQuery::create()->count(); // 2
BookQuery::create()
    ->filterByTitle('Harry Potter and the%')
    ->('Title' => 'Voldemort got you Pwned'));
```

```
}
//<li class="even">Harry Potter and the Philosopher's Stone</li>
//<li class="odd">Harry Potter and the Chamber of Secrets</li>
unset($books[1]);
unset($books[0]);
$books []= $book1;

// Query termination methods
echo BookQuery::create()->count(); // 2
BookQuery::create()
  ->filterByTitle('Harry Potter and the%')
  ->update(array('Title' => 'Voldemort got you Pwned'));
AuthorQuery::create()
  ->filterByLastName('Rowling')
  ->delete();
echo BookQuery::create()->count(); // 0

// Identity Map
$author = new Author();
$author->setFirstName('Jane');
$author->setLastName('Austen');
$author->save();
$pk = $author->getId();
$book = new Book();
```

```php
  ->update(array('Title
AuthorQuery::create()
  ->filterByLastName('Rowling')
  ->delete();
echo BookQuery::create()->count(); // 0

// Identity Map
$author = new Author();
$author->setFirstName('Jane');
$author->setLastName('Austen');
$author->save();
$pk = $author->getId();
$book = new Book();
$book->setTitle('Sense and Sensibility');
$book->setAuthorId($pk);
$book->save();
$author = $book->getAuthor(); // no query
$author = AuthorQuery::create()->findPk($pk); // no query

// On-Demand Hydration
for ($i=0; $i < 1000; $i++) {
  $book = new Book();
  $book->setTitle('book #' . $i);
  $book->save();
}
                    ::create()
```

```php
$book->setAuthorId($pk);
$book->save();
$author = $book->getAuthor(); // no query
$author = AuthorQuery::create()->findPk($pk); // no query

// On-Demand Hydration
for ($i=0; $i < 1000; $i++) {
  $book = new Book();
  $book->setTitle('book #' . $i);
  $book->save();
}
$books = BookQuery::create()
  ->setFormatter(ModelCriteria::FORMAT_ON_DEMAND)
  ->find(); // no hydration yet
foreach ($books as $book) {
  echo $book->getTitle(), "\n"; // on demand hydration
}
BookQuery::create()->forceDeleteAll();

// Validators
$book1 = new Book();
$book1->setTitle('This title is unique');
echo $book1->validate(); // true
```

```php
  ->find(); // no hydration
foreach ($books as $book) {
  echo $book->getTitle(), "\n"; // on demand hydration
}
BookQuery::create()->forceDeleteAll();

// Validators
$book1 = new Book();
$book1->setTitle('This title is unique');
echo $book1->validate(); // true
$book1->save();
$book2 = new Book();
$book2->setTitle('This title is unique');
if ($book2->validate()) { // false
  $book2->save();
} else {
  foreach($book2->getValidationFailures() as $failure) {
    echo $failure->getMessage() , "\n"; // Book title already in data
  }
}

// Behaviors
// soft delete a book
$book1->delete();
echo $book1->isDeleted(); // false
echo $book1->getDeletedAt(); // 2010-10-29 18:14:23
echo BookQuery::create()->count(); // 0
// recover a deleted book
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<database name="bookstore" defaultIdMethod="native"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.propelorm.org/xsd/1.5/database.xsd">

  <table name="book" namespace="Bookstore">
    <column name="id" required="true" primaryKey="true" autoIncrement="true" type="INTEGE
    <column name="title" type="VARCHAR" required="true" primaryString="true" />
    <column name="author_id" required="false" type="INTEGER" />
    <foreign-key foreignTable="author" onDelete="CASCADE">
      <reference local="author_id" foreign="id" />
    </foreign-key>
    <validator column="title" translate="none">
      <rule name="unique" message="Book title already in database." />
      <rule name="minLength" value="3" message="Book title must be more than ${value} cha
      <rule name="maxLength" value="255" message="Book title must not be longer than ${va
    </validator>
    <behavior name="soft_delete" />
  </table>

  <table name="author" namespace="Bookstore">
    <column name="id" required="true" primaryKey="true" autoIncrement="true" type="INTEGE
    <column name="first_name" required="true" type="VARCHAR" size="128" />
    <column name="last_name" required="true" type="VARCHAR" size="128" />
    <validator column="last_name" translate="none">
      <rule name="required" message="The last name field is required." />
    </validator>
  </table>

  <table name="content">
    <column name="id" required="true" primaryKey="true" autoIncrement="true" type="INTEGE
```

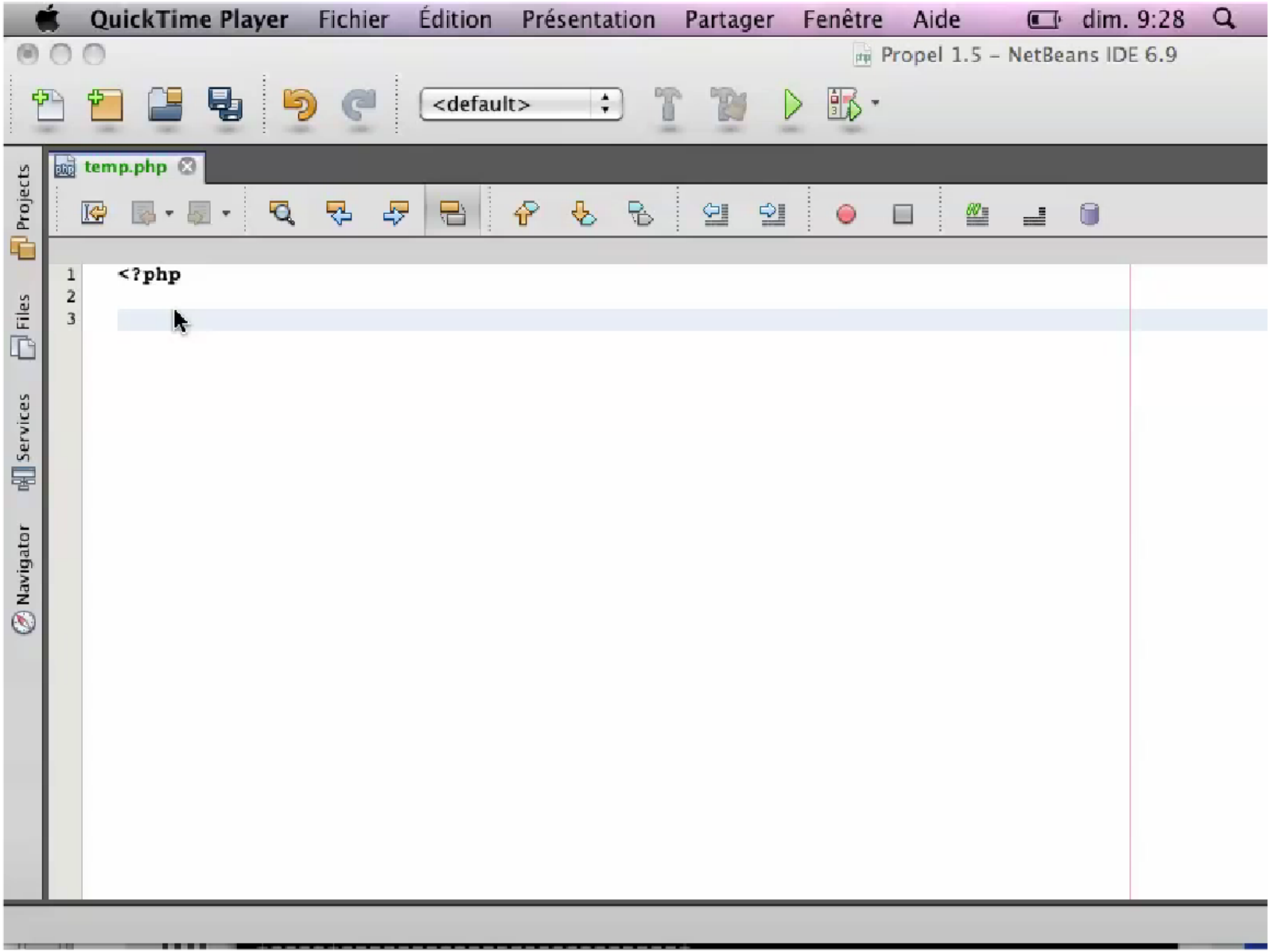
```
    }
}

// Behaviors
// soft delete a book
$book1->delete();
echo $book1->isDeleted(); // false
echo $book1->getDeletedAt(); // 2010-10-29 18:14:23
echo BookQuery::create()->count(); // 0
// recover a deleted book
BookQuery::disableSoftDelete();
echo BookQuery::create()->count(); // 1
BookQuery::disableSoftDelete();
$book = BookQuery::create()->findOne();
$book->unDelete();
BookQuery::enableSoftDelete();
echo BookQuery::create()->count(); // 1
// soft delete works on queries
BookQuery::create()->deleteAll();

// Table Inheritance
$art = new Article();
$art->setTitle('Avatar Makes Best Opening Weekend in the
$art->setIsPublished(true);
                    million worldwide total,
                    in the histor
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<database name="bookstore" defaultIdMethod="native"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.propelorm.org/xsd/1.5/database.xsd">

  <table name="book" namespace="Bookstore">
    <column name="id" required="true" primaryKey="true" autoIncrement="true" type="INTEGE
    <column name="title" type="VARCHAR" required="true" primaryString="true" />
    <column name="author_id" required="false" type="INTEGER" />
    <foreign-key foreignTable="author" onDelete="CASCADE">
      <reference local="author_id" foreign="id" />
    </foreign-key>
    <validator column="title" translate="none">
      <rule name="unique" message="Book title already in database." />
      <rule name="minLength" value="3" message="Book title must be more than ${value} cha
      <rule name="maxLength" value="255" message="Book title must not be longer than ${va
    </validator>
    <behavior name="soft_delete" />
  </table>

  <table name="author" namespace="Bookstore">
    <column name="id" required="true" primaryKey="true" autoIncrement="true" type="INTEGE
    <column name="first_name" required="true" type="VARCHAR" size="128" />
    <column name="last_name" required="true" type="VARCHAR" size="128" />
    <validator column="last_name" translate="none">
      <rule name="required" message="The last name field is required." />
    </validator>
  </table>

  <table name="content">
    <column name="id" required="true" primaryKey="true" autoIncrement="true" type="INTEGE
```

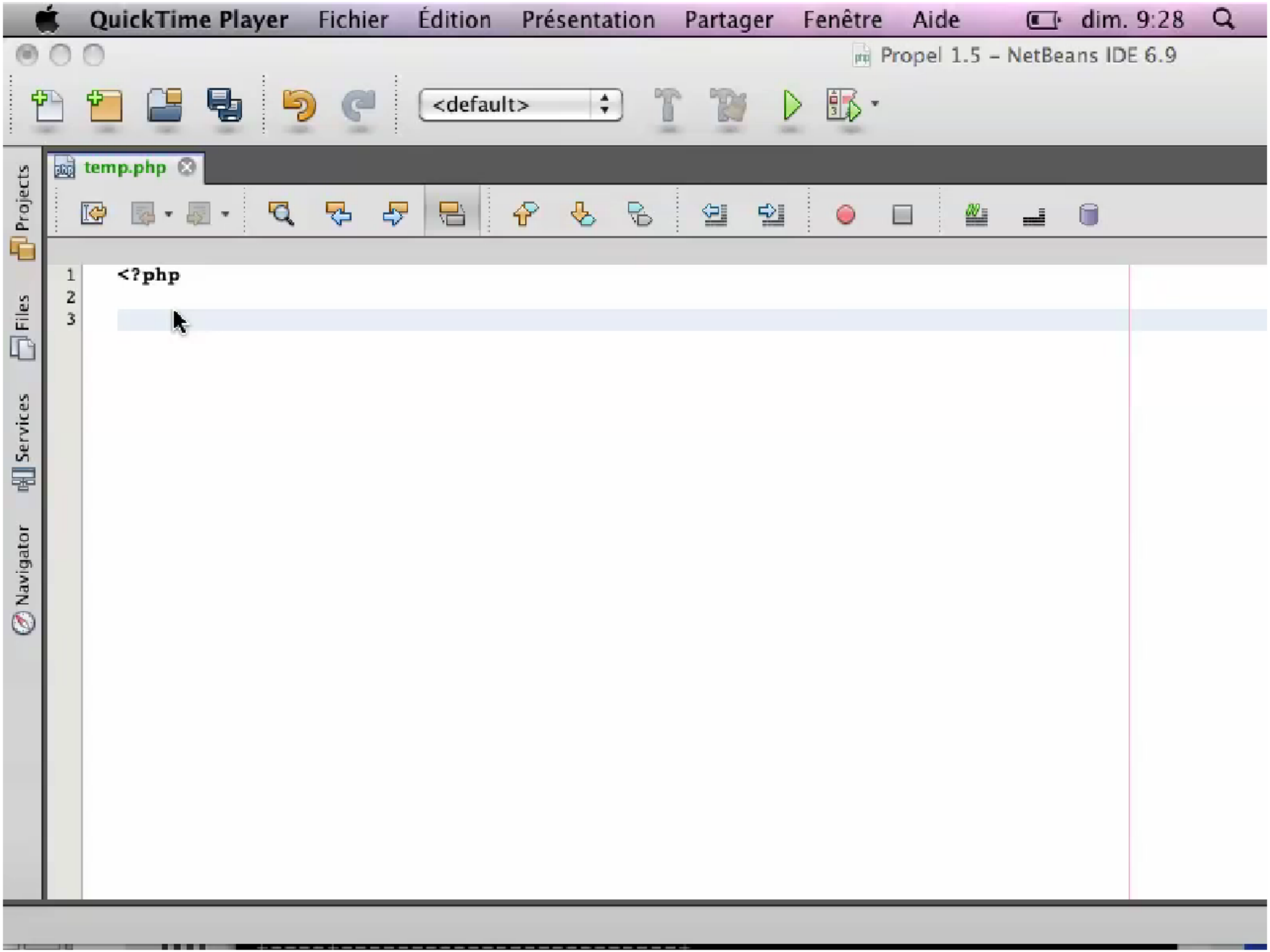
```php
BookQuery::enableSoftDelete();
echo BookQuery::create()->count(); // 1
// soft delete works on queries
BookQuery::create()->deleteAll();

// Table Inheritance
$art = new Article();
$art->setTitle('Avatar Makes Best Opening Weekend in the History');
$art->setIsPublished(true);
$art->setBody("With $232.2 million worldwide total,
Avatar had one of the best-opening weekends in the history of cinema.");
$art->save();
// create a new Video
$vid = new Video();
$vid->setTitle('Avatar Trailer');
$art->setIsPublished(true);
$vid->setUrl('http://www.avatarmovie.com/index.html');
$vid->save();

$contents = ContentQuery::create()->find();
foreach ($contents as $content) {
  echo $content->getTitle(), "\n";
}
// Avatar Makes Best Opening Weekend in the History
// Avatar Trailer
```

```
    <validator column="last_name" translate="none">
      <rule name="required" message="The last name field is required." />
    </validator>
  </table>

  <table name="content">
    <column name="id" required="true" primaryKey="true" autoIncrement="true" t
    <column name="title" type="VARCHAR" required="true" primaryString="true" /
    <column name="is_published" type="BOOLEAN" default="false" />
  </table>

  <table name="article">
    <column name="body" type="LONGVARCHAR" />
    <behavior name="concrete_inheritance">
      <parameter name="extends" value="content" />
    </behavior>
  </table>

  <table name="video">
    <column name="url" type="VARCHAR" size="255" />
    <behavior name="concrete_inheritance">
      <parameter name="extends" value="content" />
    </behavior>
  </table>

</database>
```

Autocompletion



Autocomplétion



Mais aussi

- Full Query logging
- Hooks
- Nested Sets
- Many-to-Many and One-to-one relationships
- Reverse engineering
- Runtime introspection
- Packages
- Nested transactions
- Master-slave
- Pager utility
- Query cache
- Multi-column foreign keys

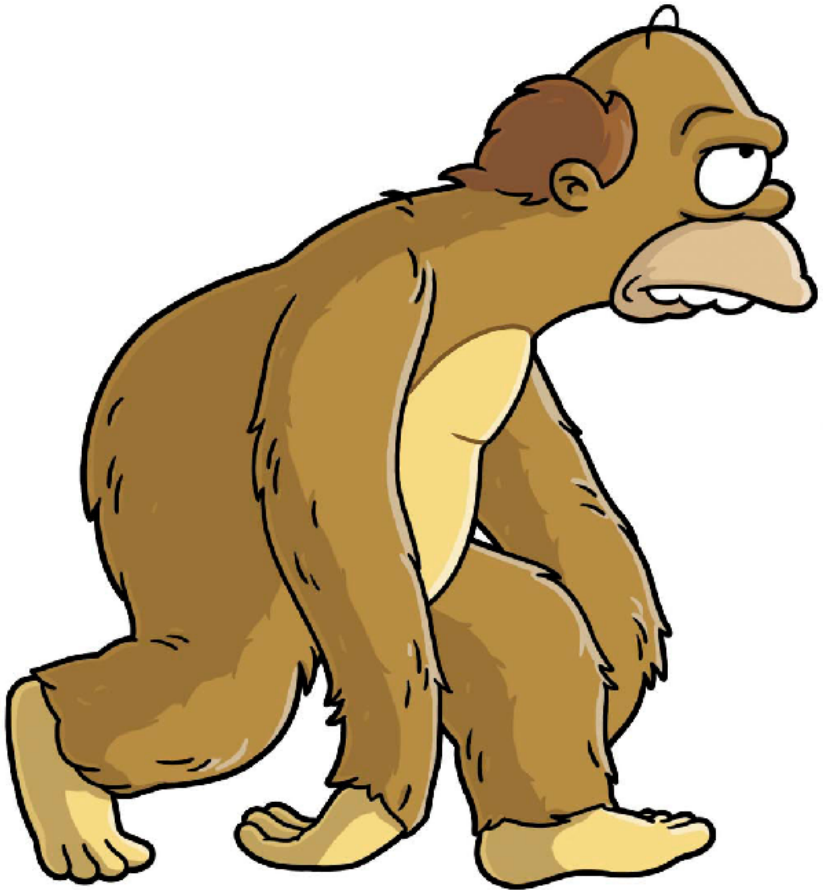
- Nested transactions
- Master-slave
- Pager utility
- Query cache
- Multi-column foreign keys
- Cascade emulation for ON DELETE and ON UPDATE
- Indexes
- Lazy-loaded columns
- LOB Support
- Views
- Sequences

- Views
- Sequences

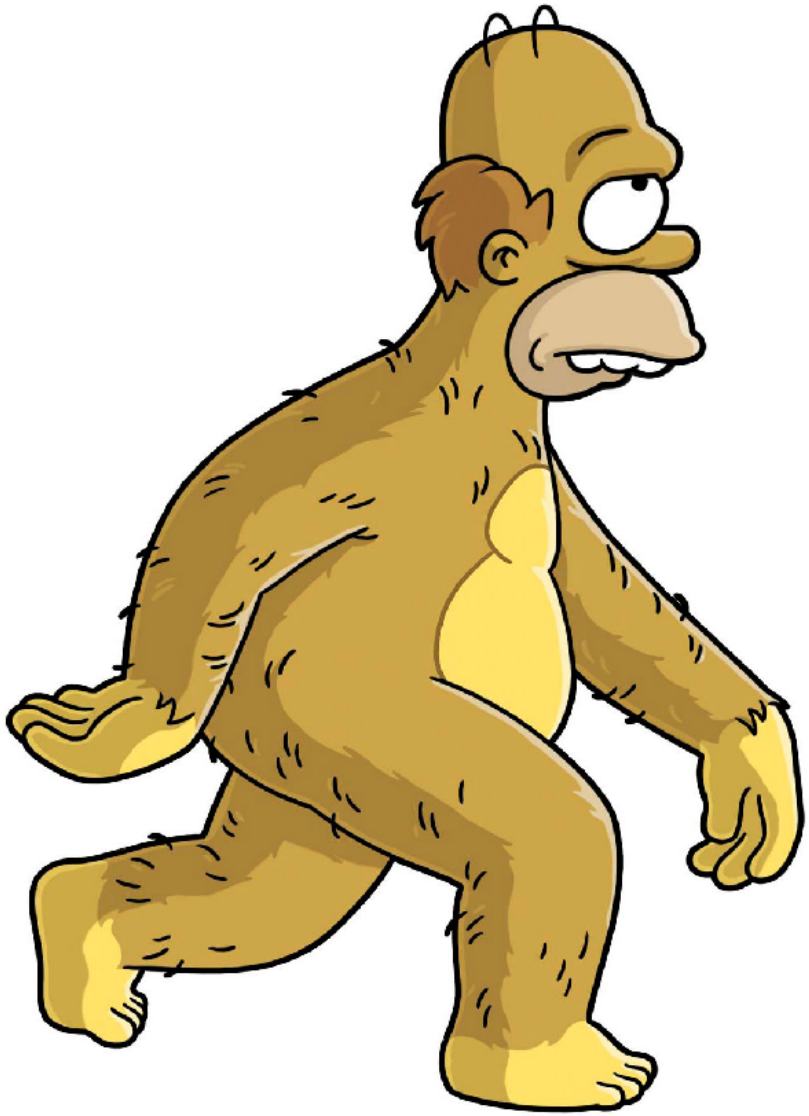
Et bientot (1.6)

- Migrations
- XML, JSON, YAML and CSV parser and dumper
- Joins with multiple conditions
- Support for Database Schemas

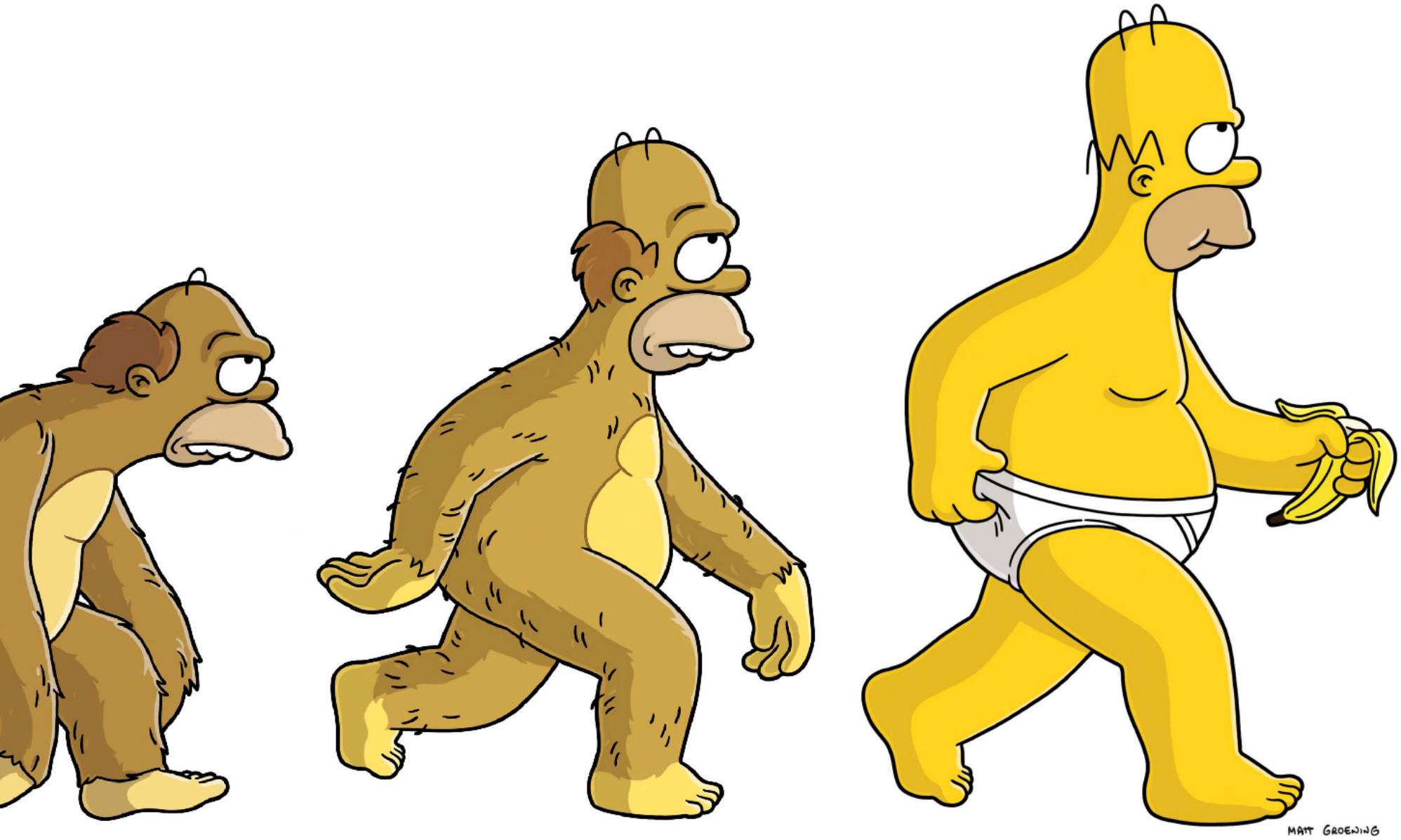
2
bjets



Niveau 3 : Ne pas
réinventer la roue



Niveau 4
Maitriser l'outil

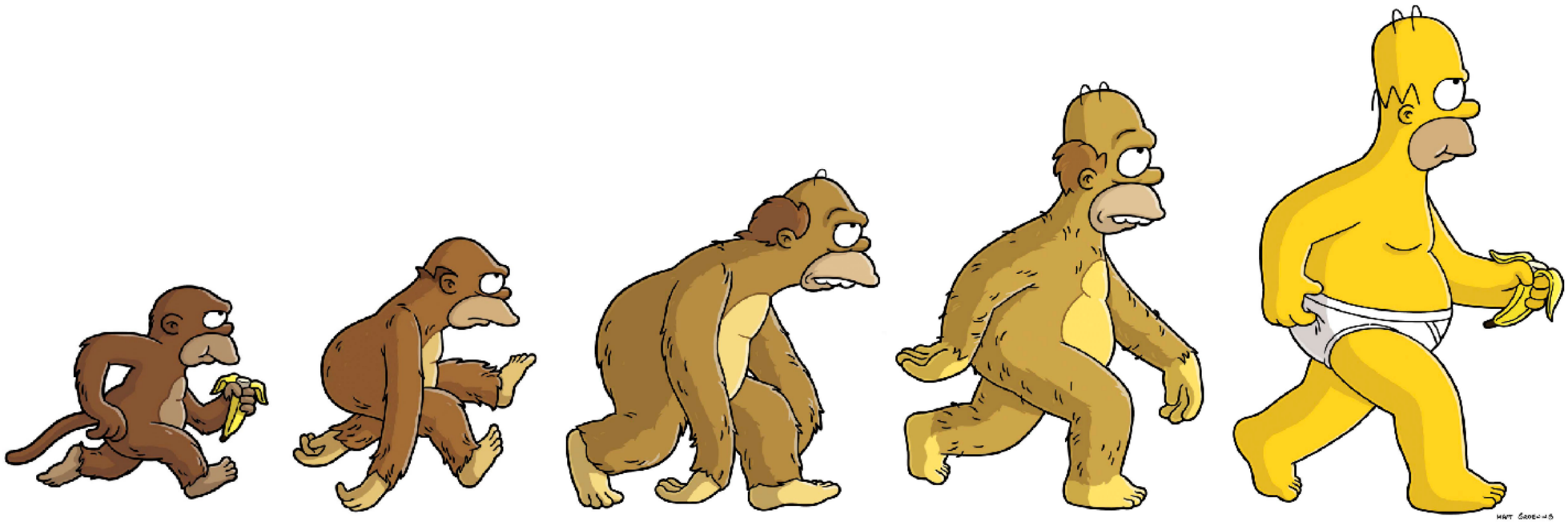


3 : Ne pas
ter la roue

Niveau 4
Maitriser l'outil

Niveau 5
Dépasser l'outil

Apprendre en persistant



Niveau 1 Niveau 2 Niveau 3 : Ne pas Niveau 4 Niveau 5