

Lógica de negocio y acceso a datos

Diseño de Sistemas Software

Curso 2022/2023

Carlos Pérez Sancho
Ana Lavalle López



Universitat d'Alacant
Universidad de Alicante

Departament de Llenguatges i Sistemes Informàtics
Departamento de Lenguajes y Sistemas Informáticos

- Arquitectura en Capas
 1. **Presentación:** interacción con el usuario
 2. **Servicios:** funcionalidades de alto nivel
 3. **Lógica de negocio:** ejecución de las reglas de negocio
 4. **Persistencia (acceso a datos):** comunicación con la BBDD
 5. **Base de datos:** almacenamiento de información

- Patrones de lógica de negocio y acceso a datos
 - **Transaction script**
 - Table Data Gateway
 - **Table module**
 - Table Data Gateway
 - **Domain model**
 - ActiveRecord
 - Data Mapper

Supuesto inicial

- Queremos implementar la funcionalidad “Procesar pedido”
 - Un pedido se compone de varias líneas de pedido
 - Cada línea de pedido está asociada a un producto e indica cuántas unidades de producto se van a vender
 - Si no hay suficientes unidades del producto en stock, el pedido no se puede procesar
 - El formulario para crear el pedido permitirá buscar productos por categoría

Mockup “Crear pedido”

New order

Product

Q search

Product 1
Product 2
Product 3

3

Add

Order

#	Product	Quantity	Price	Subtotal
1	Product 71	3	15.2	45.6
2	Product 2	1	9.95	9.95

Total: 55.55

Cancel Process

Created with Balsamiq - www.balsamiq.com

¿Cómo diseñamos la lógica de negocio?

Patrones de lógica de negocio

- Tres opciones (Fowler, 2002)
 - **Transaction script**
 - **Table module**
 - **Domain model**

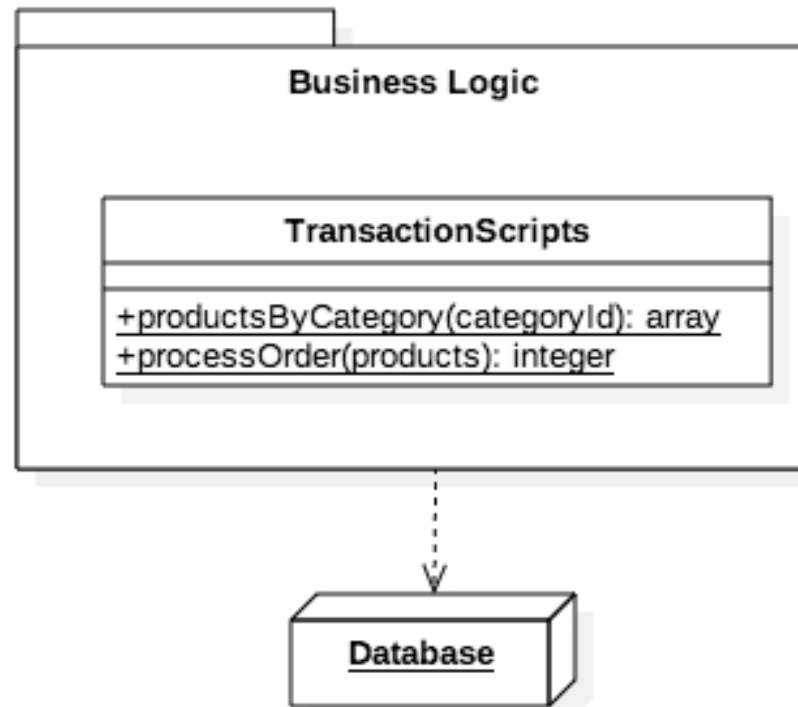
Transaction script

Patrones de lógica de negocio y acceso a datos

<https://learning.oreilly.com/library/view/patterns-of-enterprise/0321127420/ch09.xhtml#ch09lev1sec1>

Transaction Script

“Organiza la lógica de negocio en **procedimientos**, donde **cada procedimiento gestiona una petición** de la capa de presentación.”



Transaction Script

- Es la forma más **sencilla** de organizar la lógica de negocio
- Se crea una **transacción** para cada una de las funcionalidades de la aplicación
- Cada transaction script se organiza en un único método, haciendo llamadas directamente a la base de datos

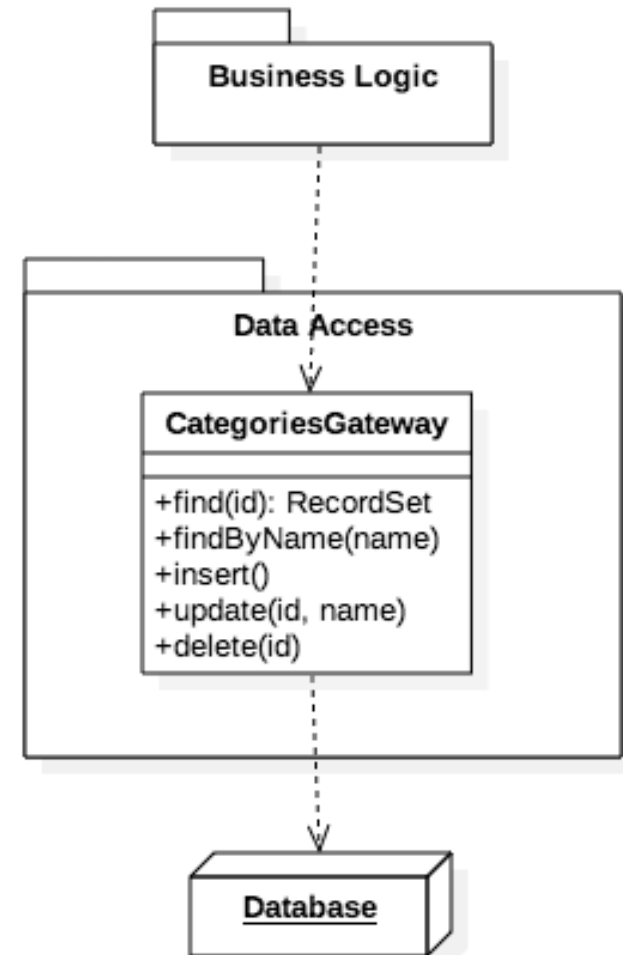
Transaction Script

```
class RawScripts {  
    public static function productsByCategory($category) {  
        $products = DB::table('products')  
            ->join('categories', 'products.category_id', '=',  
                'categories.id')  
            ->where('categories.name', $category)  
            ->select('products.id', 'products.name',  
                'products.price', 'products.stock')  
            ->get();  
        return $products;  
    }  
  
    public static function processOrder($products) {  
        // Ver proyecto en GitHub  
    }  
}
```

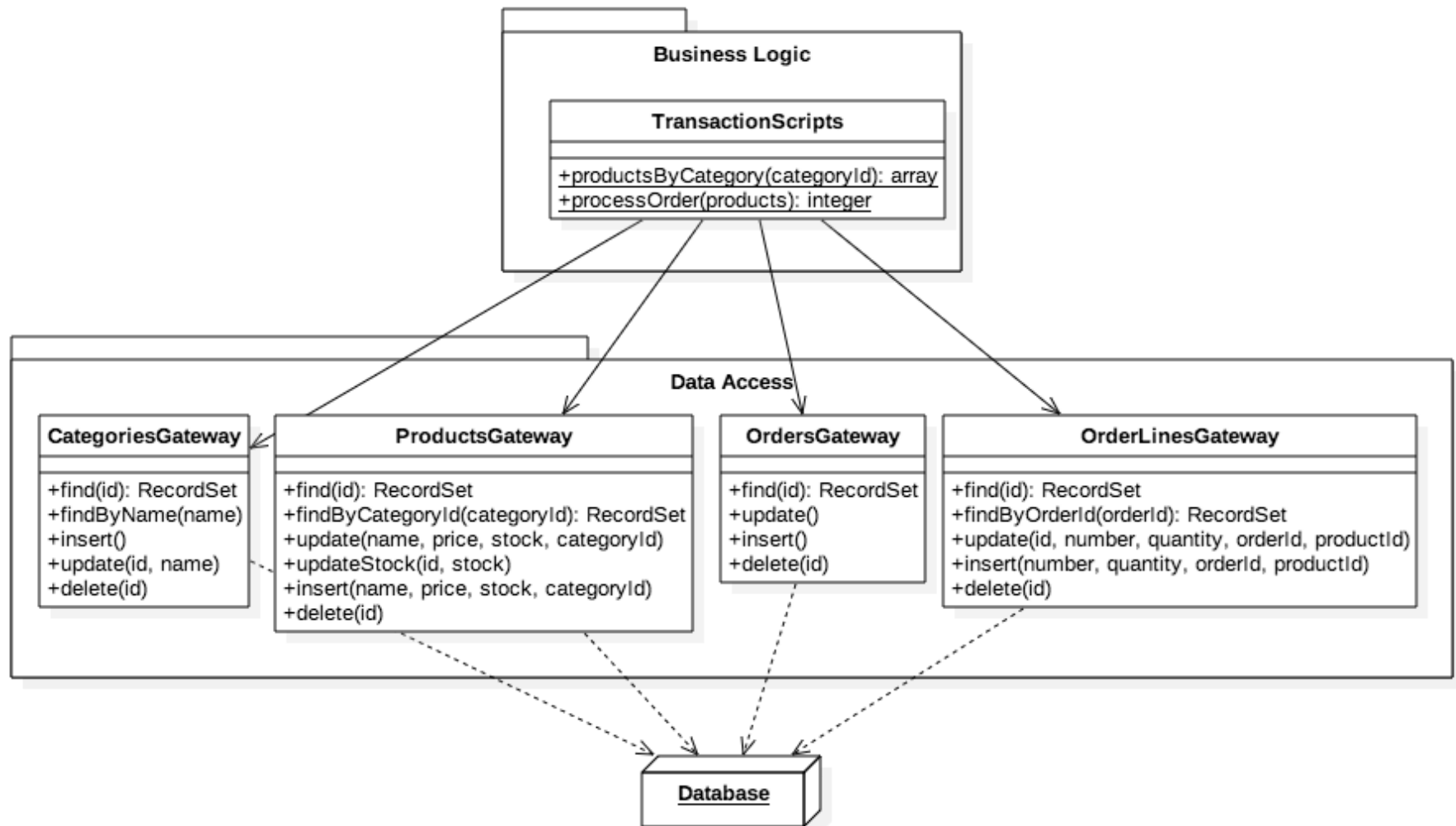
Table Data Gateway

Si queremos evitar hacer llamadas SQL directamente desde los transaction script, podemos combinarlos con el **patrón de acceso a datos** Table Data Gateway:

“Un objeto que actúa como pasarela para una tabla de la base de datos. Una única instancia gestiona todas las filas de la tabla.”



Transaction Script + Table Data Gateway



Transaction Script

```
use App\DataAccessLayer\TableDataGateways\CategoriesGateway as CG;  
use App\DataAccessLayer\TableDataGateways\OrdersGateway as OG;  
use App\DataAccessLayer\TableDataGateways\OrderLinesGateway as OLG;  
use App\DataAccessLayer\TableDataGateways\ProductsGateway as PG;
```

```
class Scripts {  
    public static function productsByCategory($category) {  
        $category = CG::findByName($category);  
        if ($category)  
            return PG::findByCategoryId($category->id);  
        else  
            return null;  
    }  
  
    public static function processOrder($products) {  
        // View project at GitHub  
    }  
}
```

Transaction Script - Recomendaciones

- **Recomendado** para aplicaciones con poca lógica de negocio
 - Implica poca sobrecarga, ya sea en rendimiento o en comprensión
- **No recomendado** cuando la lógica de negocio se vuelve más complicada
 - Puede aparecer duplicación entre transacciones. Como se busca manejar cada transacción, cualquier código común tiende a estar duplicado

Table module

Patrones de lógica de negocio y acceso a datos

<https://learning.oreilly.com/library/view/patterns-of-enterprise/0321127420/ch09.xhtml#ch09lev1sec3>

Table Module

“Una **única instancia** gestiona la lógica de negocio **para** todas las filas en una **tabla** o vista de la base de datos.”

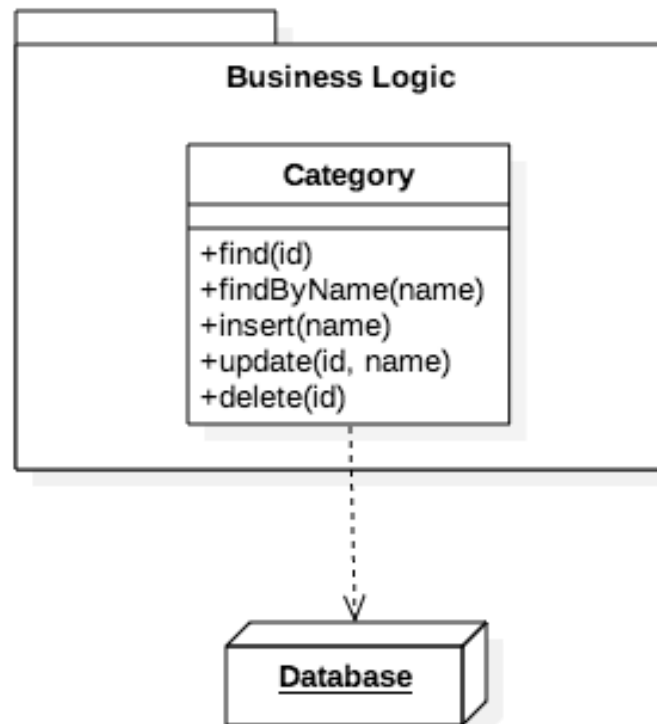


Table Module

- Una **única instancia** del objeto Table Module permite gestionar todos los registros de la tabla
- Todos los métodos excepto `insert()` necesitan un **identificador** para localizar el registro en la base de datos
- Los objetos table module también pueden usar el **patrón Table Data Gateway** para comunicarse con la base de datos
 - Table Module implementa la lógica de negocio
 - Table Data Gateway implementa el acceso a datos

Table Module + Table Data Gateway

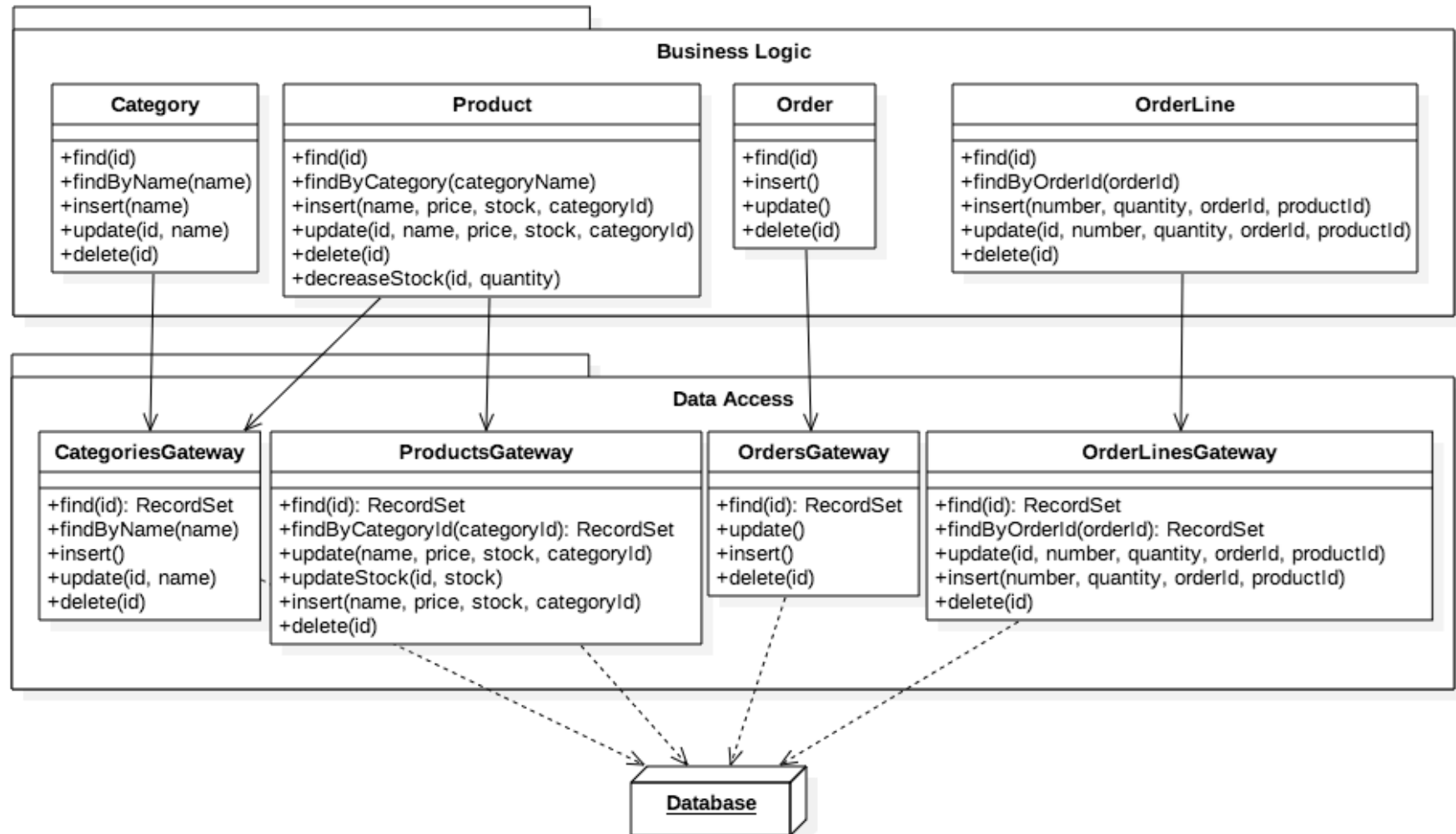


Table Module

```
use App\DataAccessLayer\TableDataGateways\ProductsGateway as PG;

class Product {

    public static function findByCategory($category) {
        $category = Category::findByName($category);
        if ($category) return PG::findByCategoryId($category->id);
        else return null;
    }

    public static function decreaseStock($id, $quantity) {
        $product = PG::find($id);
        if ($product && $product->stock >= $quantity) {
            PG::updateStock($id, $product->stock - $quantity);
            return true;
        }
        else return false;
    }

    // View project at GitHub
}
```

- Los objetos Table Module sólo contienen la lógica de negocio que corresponde a cada objeto individual
- Las operaciones complejas deben ir en la **capa de servicio**

Table Module + Capa de servicio

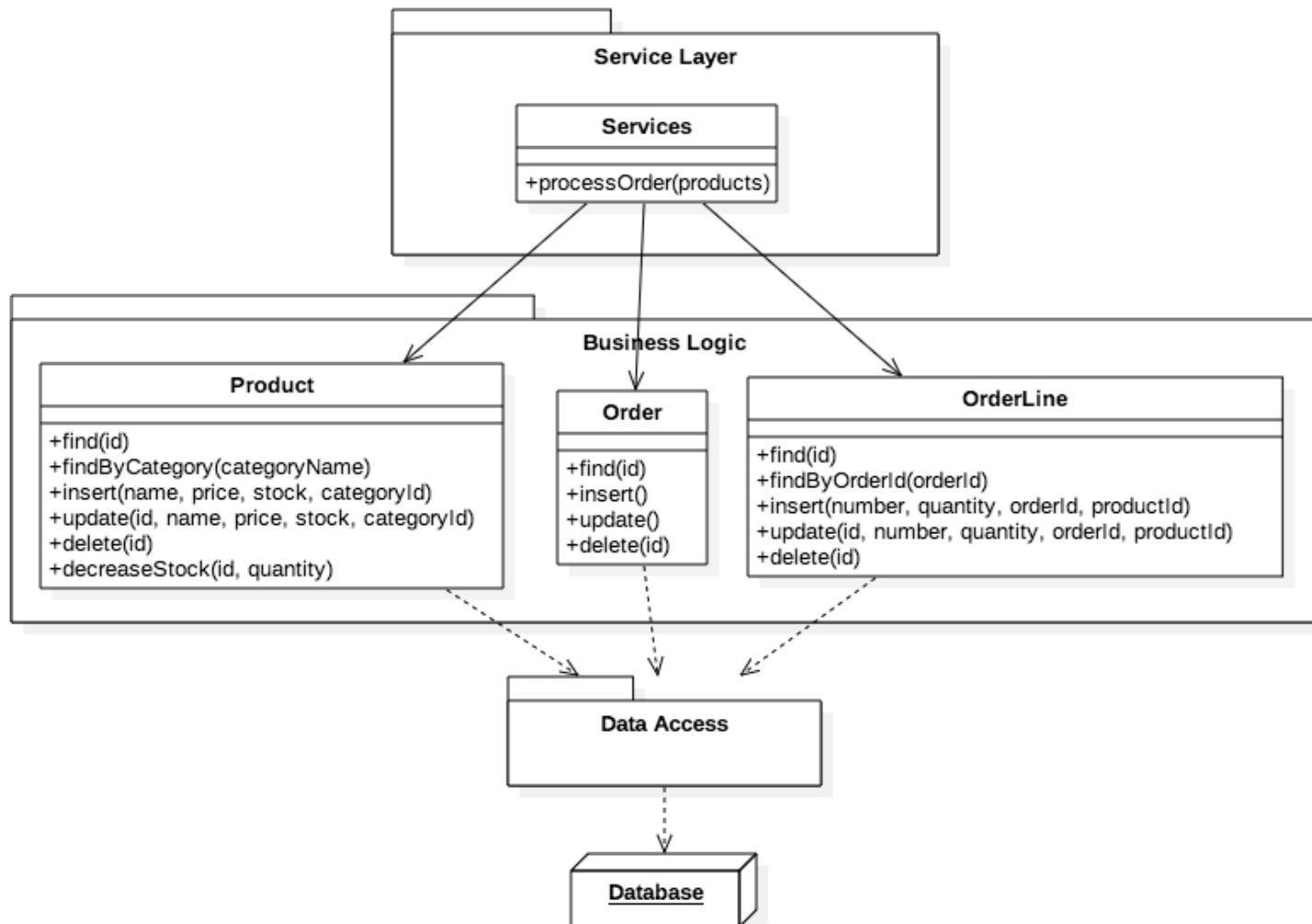


Table Module

```
use App\BusinessLogicLayer\TableModules\Order;
use App\BusinessLogicLayer\TableModules\OrderLine;
use App\BusinessLogicLayer\TableModules\Product;

class TableModuleServices {
    public static function processOrder($products) {
        $rollback = false;
        $lineNumber = 1;

        DB::beginTransaction();

        $orderId = Order::insert();
        foreach ($products as $productId => $quantity) {
            if (Product::decreaseStock($productId, $quantity)) {
                OrderLine::insert($lineNumber, $quantity, $orderId,
                                $productId);

                $lineNumber++;
            }
        }

        // View project at GitHub
    }
}
```

Domain model

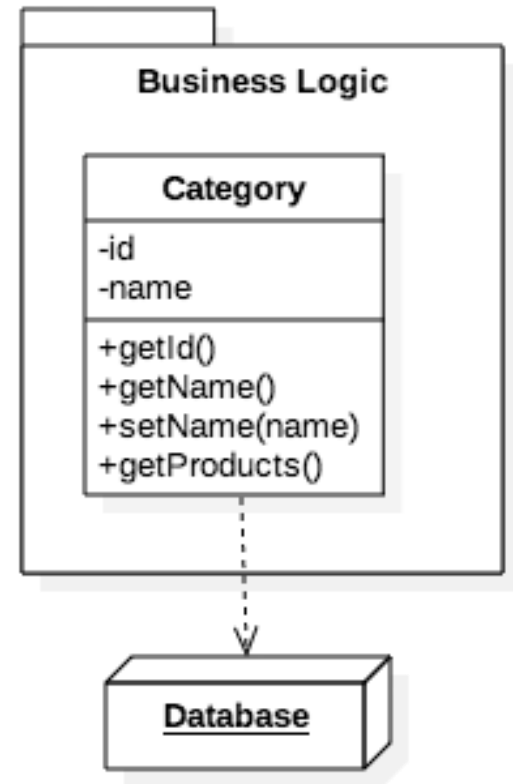
Patrones de lógica de negocio y acceso a datos

<https://learning.oreilly.com/library/view/patterns-of-enterprise/0321127420/ch09.xhtml#ch09lev1sec2>

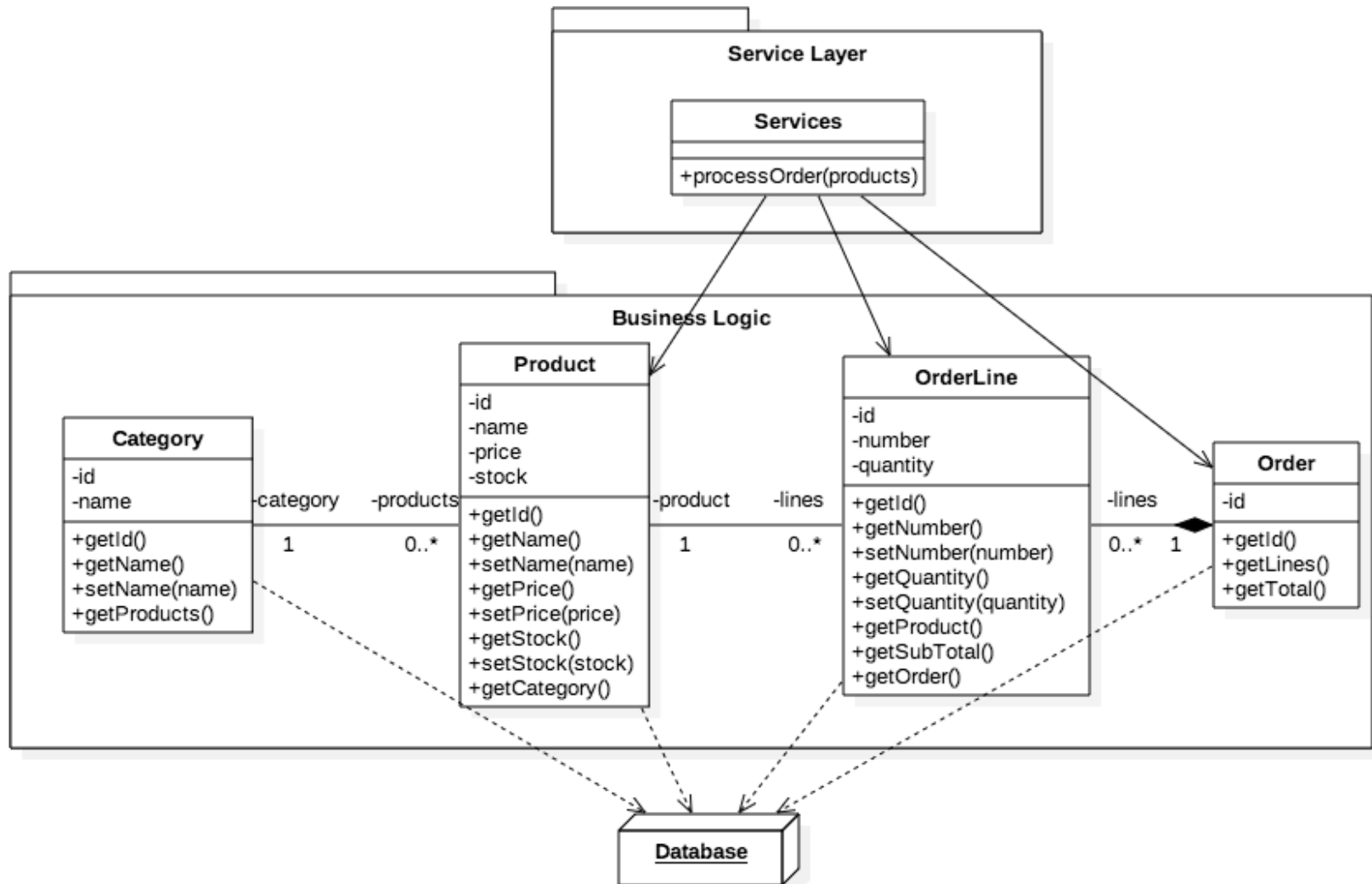
Domain Model

Cuando la **lógica de negocio es compleja** la mejor opción es implementar un modelo de dominio:

*“Un objeto del modelo de dominio incorpora tanto el **comportamiento** como los **datos**.”*



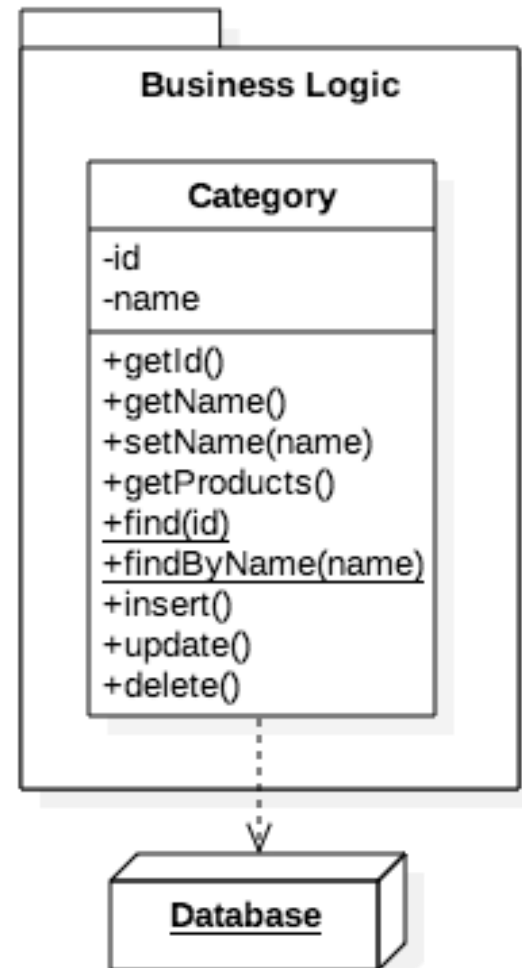
Domain Model



Domain Model

- Transformar el modelo de dominio a una estructura de tablas relacional puede ser complejo
 - Los modelos sencillos tienen un objeto por cada tabla
 - Modelos más complejos pueden tener una estructura distinta a la de la base de datos
- Dependiendo de la complejidad el acceso a datos se puede hacer de dos maneras
 - Modelos sencillos → **ActiveRecord**
 - Modelos complejos → **Data Mapper**

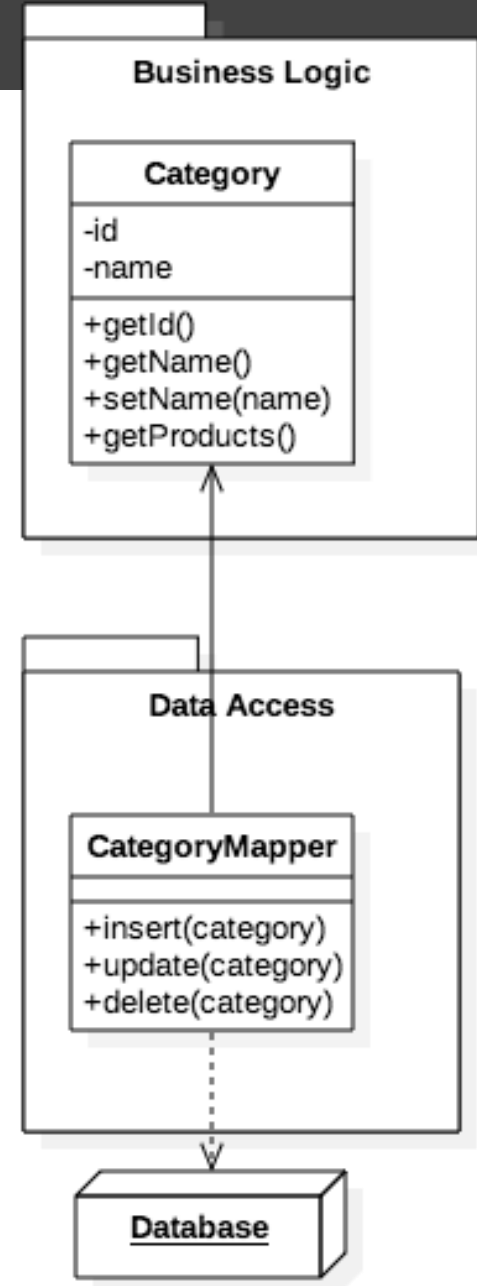
*“Un **objeto** que envuelve una fila de una tabla o vista, **encapsula el acceso** a la base de datos y **añade comportamiento** a esos datos.”*



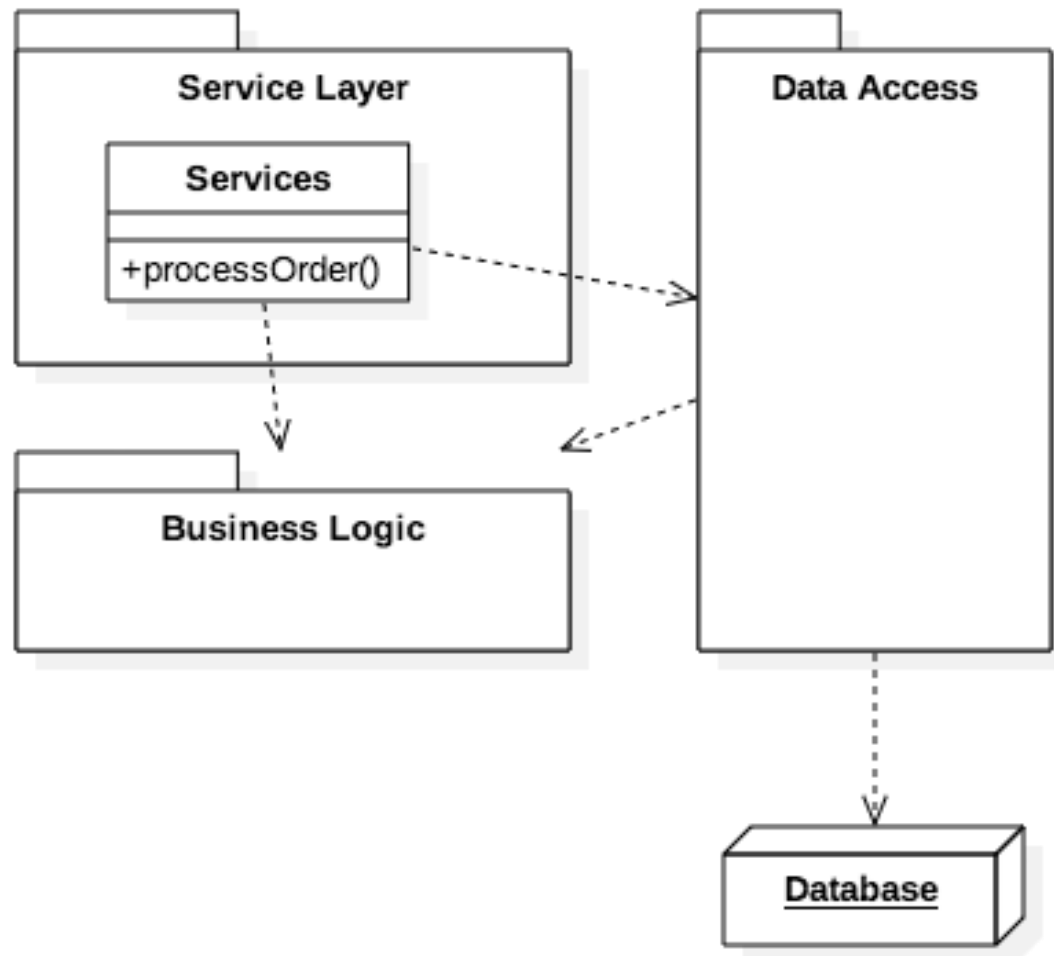
Data Mapper

Si **modelo de dominio** es complejo conviene mantenerlo **separado** del **acceso a datos**. En estos casos se usa el patrón Data Mapper:

“Una capa de mapeadores se encarga de mover datos entre los objetos y la base de datos, manteniéndolos independientes entre ellos e independientes del mapeador.”




Data Mapper



Resumen

Patrones de lógica de negocio y acceso a datos

Resumen

Complejidad	Lógica de negocio	Acceso a datos
	Transaction Script	-
		Table Data Gateway
	Table Module	-
		Table Data Gateway
	Domain Model	ActiveRecord
		Data Mapper

Ejercicios

Patrones de lógica de negocio y acceso a datos

<https://github.com/cperezs/dss-business-logic-patterns/tree/master/app>

- Revisar el código y realizar una traza:
 - App\BusinessLogicLayer\TransactionScripts\Scripts
 - App\ServicesLayer\TableModuleServices
 - App\ServicesLayer\ActiveRecordServices

¿Preguntas?

- Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley Professional.
[Leer en O'Reilly Online](#)