**Magdalena Borys**[1], **Beata Panczyk**[2]

## IMPROVING DATA PROCESSING PERFORMANCE FOR WEB APPLICATIONS USING OBJECT-RELATIONAL MAPPING

***The paper presents the object-relational mapping (ORM) technology as one of solution for increasing system flexibility and efficiency during the whole system life cycle. ORM frameworks not only fill the gap between the object model and the relational database, but also facilitate the work of web developers and decrease the cost of website maintenance. Increasing the efficiency of performing tasks on data is shown using the simple web application. The results are presented as the summary statements. The paper focuses on ORM technologies in the context of data performance efficiency. It also presents other issues associated with Hibernate, LINQ and Doctrine frameworks.***
***Keywords:*** *efficiency of data processing; object-relational mapping; web application.*

**Магдалена Борис, Беата Панчик**

## ПІДВИЩЕННЯ ЯКОСТІ ОБРОБКИ ДАНИХ ДЛЯ ВЕБ-ПРИКЛАДОК З ВИКОРИСТАННЯМ ОБ'ЄКТНО-РЕЛЯЦІЙНОГО ВІДОБРАЖЕННЯ

***У статті описано технологію об'єктно-реляційного відображення (ОРВ) як один зі способів підвищення гнучкості та ефективності всієї системи. ОРВ не тільки заповнює пробіл між самим об'єктом та базою даних, а також прискорює роботу девелоперів та зменшує витрати на підтримку веб-сайтів. Підвищення ефективності виконання задач продемонстровано на прикладі простої веб-прикладки. ОРВ-технології особливо актуальні в контексті підвищення ефективності даних. Коротко описано принципи роботи Hibernate, LINQ та Doctrine.***
***Ключові слова:*** *ефективність обробки даних; об'єктно-реляційне відображення; веб-прикладка.*
***Рис. 3. Табл. 6. Літ. 10.***

**Магдалена Борис, Беата Панчик**

## ПОВЫШЕНИЕ КАЧЕСТВА ОБРАБОТКИ ДАННЫХ ДЛЯ ВЕБ-ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ ОБЪЕКТНО-РЕЛЯЦИОННОГО ОТОБРАЖЕНИЯ

***В статье описана технология объектно-реляционного отображения (ОРО) как один из способов повышения гибкости и эффективности всей системы. ОРО не только заполняет пробел между самим объектом и базой данных, но также ускоряет работу девелоперов и снижает расходы на поддержку веб-сайта. Повышение эффективности выполнения задач показано на примере простейшего веб-приложения. ОРО-технологии особенно актуальны в контексте повышения эффективности данных. Кратко описаны принципы работы Hibernate, LINQ и Doctrine.***
***Ключевые слова:*** *эффективность обработки данных; объектно-реляционного отображение; веб-приложения.*

**The basics of ORM.** Almost every application must store data and almost all large, contemporary applications are written in the object way. But data are relational by nature and databases store them in a relational way. Object orientation for applications and relation of databases are natural. But it is much faster to write object systems and easier to design and divide tasks. Relational databases seem to be irreplaceable. They are very effective. Introducing another model of storing data for now is not

[1] Lublin University of Technology, Poland.
[2] Lublin University of Technology, Poland.

acceptable. However, it is possible to combine in one system both object orientation of its structure and relational model of storing data (Barry, 2012).

Data are downloaded from a database many times, transferred to an application and saved again after converting by a system. It is possible to do it using SQL and programming languages (Java, C#, PHP etc.).

In an application usually there are defined classes whose objects are "boxes" to store data for a short period of time as well as to pass on information between various layers and objects of layers. Fields of objects are usually the same as the fields of tables storing tuples describing these objects in a database (Hibernate, 2013).

**Transparent persistence.** Meaning of persistence is very important in application development. Almost all applications require persistent data. It is very useful for a system to preserve data entered by users.

Transparent persistence in ORM products is the ability to directly manipulate data stored in a relational database using an object programming language. A database sub-language uses the embedded SQL and for example ODBC (Open Database Connectivity) or JDBC (Java Database Connectivity) calls the appropriate interface.

With transparent persistence, manipulation of persistent objects is performed directly by the object programming language in the same way as in the memory, non-persistent objects. This is achieved through the use of intelligent caching (Barry, 2012).

**Caching data.** Caching is used for storing data in the application to minimize network traffic and disk access. Caching and transparent persistence is often set up as a part of application work space. With cache, the application doesn't have to explicitly translate relational tuples. They move from disk storage automatically into program memory. If client's applications read the same data multiple times then caching can help improve performance. The development cost of using RDBMS with objects is reduced because object-relational mapping products take care of the object-to-table and table-to-object conversion. Without an object-relational mapping product, this conversion would need to be written in addition to application development.

Data in the cache must be synchronized with the data in the database managed by DBMS server. A cache is used by an application that is separate from the underlying DBMS server (Figure 1).

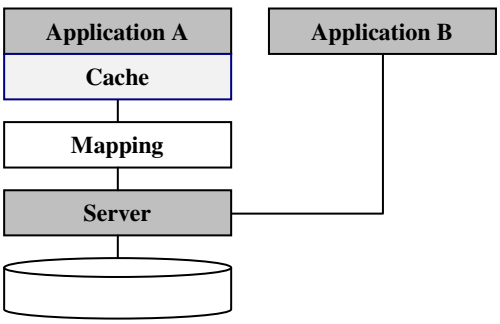


*Figure 1.* **Using a cache with object relational mapping** (Barry, 2012)

Figure 1 shows the situation when the second application *B* is accessing the same DBMS server that is being used by an object application *A*. Application *A* uses a cache

with object-relational mapping. Application *B* can change the data used by Application *A* and cache *A* would need to be synchronized to obtain the changed data.

**Object to table mapping.** Mapping objects to tables involves creating or updating data stored in a relational database. Mapping involves (Barry, 2012):

- mapping objects to one or many tables – there are multiple ways to map objects to tables – mapping inheritance issues and determining how many classes per table;
- making decisions related to potential for redundant data;
- designing for multi-table updates;
- mapping collection classes into tables – a collection is mapped to one or more tables;
- mapping object types to database data types – in most cases this is straightforward mapping;
- mapping object relationships to keys or intersection tables – this is the many-to-many relationship issue represented in a relational database as an intersection table.

**Table to object mapping.** Using the existing relational data with objects at the application requires mapping tables to objects. Mapping involves (Barry, 2012):

- mapping relational data types into object types;
- mapping relational table definitions to object classes – the table definitions map directly except for foreign keys, which are replaced by relationships;
- mapping inheritance based on a table or multiple tables;
- mapping tuple retrieval, keys and relational joins to relationships for object navigation;
- mapping "intersection tables" to object relationships.

Figure 2 illustrates the table-object mapping.

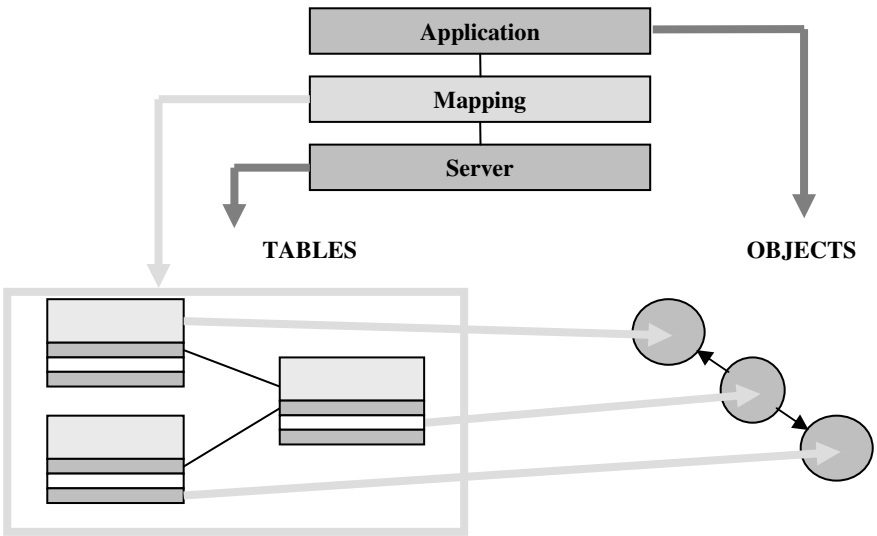


*Figure 2.* **Table-object mapping** (Barry, 2012)

**ORM technologies.** Depending on the web application development environment, 3 most popular ORM technologies can be identified: Hibernate for Java, LINQ for ASP and Doctrine for PHP. In this section they are briefly described.

**Hibernate.** Hibernate is an open source ORM mapping library for the Java language, that provides persistent classes and logic without caring how to handle data (Hibernate, 2013a; Hibernate, 2013b). Persistence in Java (Hibernate) is storing data in a relational database using SQL.

Hibernate let us develop persistent classes following object-oriented idiom (association, inheritance, polymorphism, composition, collections). Hibernate allows expressing queries in native SQL or in its own portable SQL extension (HQL) or with an object-oriented criteria and example API. This can reduce the development time. Programmer doesn't have to handle the data manually in SQL and JDBC.

The most important Hibernate's feature is mapping from Java classes to database tables and from Java data types to SQL data types.

Hibernate allows creating high-performance database applications with Java much faster and easier. Transparent persistence for Plain Old Java Objects (POJOs) allow building a simple POJO, then create XML mapping file that will describe the relationship between the database and the class attributes and at the end call Hibernate API's to specify operations (Figure 3). A programmer may not be aware when and which queries are performed to a database (Hibernate, 2013a).

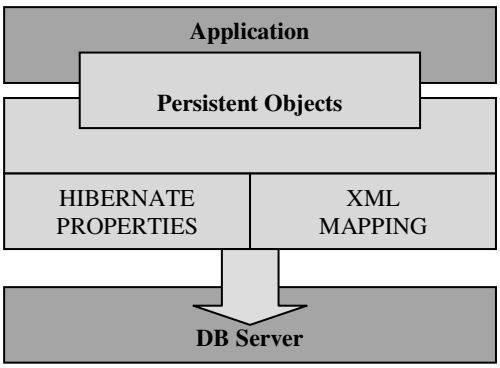


*Figure 3.* **Hibernate performance patten** (Hibernate, 2013a)

Hibernate can be used as in standalone Java applications or as in Java web applications (Rychlicki-Kicior, 2010; Panczyk, 2010).

Hibernate is an ORM tool that automatically maps object to the relational database and helps developers quickly write database access program and become productive. Java developers mostly use Hibernate for developing enterprise web applications. Hibernate is better than plain JDBC: it is possible to use Hibernate which generates the SQL on the fly and then automatically executes the necessary SQL statements. This saves a lot of development and debugging time in comparison with writing JDBC statement, setting the parameters, executing query and processing the result by hand. Mapping the domain object with relational database let's concentrate on a business logic rather than managing data in a database. It's database independent. There are also many caching frameworks that work with Hibernate. It is possible to use any of them to improve the performance of an application.

The problem with Hibernate is that a lot of effort is required to learn it. Sometimes debugging and performance tuning becomes difficult. Hibernate is slo-

wer than pure JDBC as it is generating lots of SQL statements in runtime. It advisable to use pure JDBC for batch processing.

**LINQ** (Language Integrated Query) is a Microsoft programming model and methodology that gives a formal query capabilities into Microsoft .NET-based programming languages. It is a set of extensions to the .Net Framework 3.5 and its managed languages that set the query as an object and define a common syntax and a programming model to query different types of data using a common language.

Relational operators (Select, Project, Join, Group, Partition, Set etc.), are implemented in LINQ and the C# and VB compilers in the .Net framework 3.5, which support the LINQ syntax make it possible to work with a configured data store without resorting to ADO.NET (LINQ, 2013). LINQ may be used to access all types of data (database, text files and XML), whereas embedded SQL is limited to addressing only databases that can handle SQL queries.

The LINQ advantages are:

- quick turnaround for development;
- queries can be dynamic;
- tables are automatically created into class, columns into properties;
- relationship are automatically appeared to classes;
- data is easy to setup and use.

But LINQ sends the entire query to the DB hence it takes much network traffic but the stored procedures sends only argument and the stored procedure name – it will become bad if queries are very complex. Performance is degraded if the LINQ query is not correct and to make changes in data access, it is necessary to recompile and redeploy the whole assembly.

**Doctrine** is ORM that provides transparent persistence for PHP objects. It uses the Data Mapper pattern at the heart of this project, aiming for a complete separation of the domain/business logic from the persistence in a relational database management system. The benefit of Doctrine for programmers is is ability to focus solely on the object-oriented business logic and persistence is only as a secondary task. This does not mean persistence is not important to Doctrine, however there are considerable benefits for object-oriented programming if persistence and entities are kept perfectly separated (Doctrine, 2013).

Actually, Doctrine is quite simple in its fundamentals. And it is a very good choice for small, medium and even some large applications. Doctrine is not the answer for everything and sometimes it is a little problematic. However, for typical tasks it is extremely useful.

**The case study** of ORM implementation was build using one of the most popular PHP open source framework Symfony (Symfony, 2013). Symfony version 2.0.16 is integrated with Doctrine 2 ORM. For the purpose of the paper the sample database "World database" structure and data was used (MySql, 2013).

The case study is presented to show how integrating ORM framework into web application development can facilitate the work of web developers providing:

- automatic schema generation;
- mapping between XML and objects;
- criteria application programming interface (API);
- security.

The instructions presented in the Listing 1 create XML mapping metadata based on existing database structure, import the mapping metadata in form of annotations and finally generate entities. Those 3 commands build date schema that allows easy and flexible data manipulation.

*Listing 1.* **Instructions for generated entities from database structure**

```
php app/console doctrine:mapping:convert xml
./src/Pl/TestBundle/Resources/config/doctrine/metadata/orm --from-database –force
php app/console doctrine:mapping:import PlTestBundle annotation
php app/console doctrine:generate:entities AcmeBlogBundle
```

The example of a generated entity is presented in the Listing 2. The entity called City contains two variables id and name. The attributes of variables such as type, length, strategy of generation are established automatically based on data structure in a database. The annotations presented in the listing show the relationship between entity elements and existing data structure. For example the code @ORM\Table(name="city") expresses that the equivalent of an entity in the database is table City. Moreover, the setter and getter methods to access private entity variables are created automatically providing easy access to entity usage.

*Listing 2.* **The example of Doctrine entity**

```
Namespace Acme\DemoBundle\Entity;
use Doctrine\ORM\Mapping as ORM;

/**
 * Acme\DemoBundle\Entity\City
 *
 * @ORM\Table(name="city")
 * @ORM\Entity
 */
class City
{
 /**
 * @var integer $id
 *
 * @ORM\Column(name="ID", type="integer", nullable=false)
 * @ORM\Id
 * @ORM\GeneratedValue(strategy="IDENTITY")
 */
 private $id;

 /**
 * @var string $name
 *
 * @ORM\Column(name="Name", type="string", length=35, nullable=false)
 */
 private $name;

 /**
 * Set name
 *
```

**Continuation of** *Listing 2*

```
* @param string $name
 */
 public function setName($name)
 {
 $this->name = $name;
 }

 /**
 * Get name
 *
 * @return string
 */
 public function getName()
 {
 return $this->name;
 }
}
```

The sample of using Doctrine entity in Symfony framework controller, responsible for an application logic, is presented in the Listings 3 and 4. As shown in the Listing 3 ORM Doctrine allows for using Doctrine Query Language similar to SQL query statements to retrieve the data objects. The Listing 4 presents the implementation of Doctrine repository methods for the similar query as presented in the Listing 3.

*Listing 3.* **The example of usage of Doctrine entity in Symfony controller**

```
public function index2Action() {
 $em = $this->getDoctrine()->getEntityManager();
 $query = $em->createQuery('SELECT c FROM PlTestBundle:City c
 WHERE c.id > :id ORDER BY c.id ASC')
 ->setParameter('id', '5')
 //->limit(1000);

 $cities = $query->getResult();

 if (!$cities) {
 throw $this->createNotFoundException('No city found');
 }

 return array('cities' => $cities);
 }
```

In both Listings 3 and 4 the examples the API criteria such as setParameter, limit or orderBy are used. Additionally, using Doctrine repository and all sorts of helpful methods like presented in the Listing 5 makes fetching an object or objects from the database even easier.

Besides the presented advantages of helping web developers in their work, ORM increases also the efficiency of web application performance and maintenance. ORM provides the mechanism for caching objects, queries and metadata and allows for lazy loading of data and thus better use of resources and more stable performance of application.

*Listing 4.* **The example of usage of Doctrine entity in Symfony controller**

```
public function index3Action() {
 $repository = $this->getDoctrine()->getRepository('PlTestBundle:City');
 $query = $repository->createQueryBuilder('c')
 ->where('c.id > :id')
 ->setParameter('id', '5')
 ->orderBy('c.id', 'ASC')
 ->getQuery();

 $cities = $query->getResult();

 if (!$cities) {
 throw $this->createNotFoundException('No city found');
 }

 return array('cities' => $cities);
 }
```

*Listing 5.* **The example of usage of Doctrine entity in Symfony controller**

```
public function index4Action() {
 $repository = $this->getDoctrine()->getRepository('AcmeDemoBundle:City');
 $cities = $repository->findAll(array('id' => 'ASC'));

 if (!$cities) {
 throw $this->createNotFoundException('No city found');
 }

 return array('cities' => $cities);
}
```

While it is not always appropriate to cache objects, it is highly recommended in a production environment to cache the transformation of a Doctrine Query Language query to its SQL counterpart and thus to improve Doctrine performance and application performance. The second recommendation is the metadata cache, that avoids having to parse class metadata coming from annotations or XML in each request.

The comparison of number of queries and the time of execution from Symfony Profiler is presented in Table 1 for each of codes presented in the Listings 3–5 without caching. The results shows that all ORM fetching object methods have similar query execution time and none of them slow the performance of web application.

*Table 1.* **Comparison of ORM code results**

| Code example | Application execution time | Select query execution time | Number of queries |
|---|---|---|---|
| Listing 3 | 1102 ms | 13.98 ms | 2 |
| Listing 4 | 1128 ms | 14.14 ms | 2 |
| Listing 5 | 1083 ms | 12.58 ms | 2 |

**Conclusion.** Object-relational mapping reduces programming code and improve application performance in comparison with classical SQL interface usage. The performance cost of using RDBMSs with objects can be significantly reduced. Most

ORM mapping products provide transparent persistence with object programming languages. Moreover, it is easier for programmers to work with objects than to create SQL queries.

**References:**

*Barry, D.K.* (2013). Object-relational-mapping and Transparent Persistence, June 17, 2013 // www.service-architecture.com.

Framework Symfony, July 23, 2013 // symfony.com.

Getting started xml edition, June 17, 2013 // Doctrine tutorial // doctrine-orm.readthedocs.org.

Hibernate Homepage, May 12, 2012 // www.hibernate.org.

Hibernate tutorial, June 17, 2012 // www.hibernatetutorial.com.

*Iyer, L.S., Gupta, B., Johri, N.* (2005). Performance, scalability and reliability issues in web applications. Industrial Management & Data Systems, 105(5): 561–576.

LINQ tutorial, June 18, 2013 // www.tutorialspoint.com.

MySQL Documentation (2013). Other MySQL Documentation, June 26, 2013 // dev.mysql.com.

*Panczyk, B.* (2010). Object-Relational Mapping for Web Applications. In: Advanced object-oriented technology. Ed. Marek Milosz. PIPS Polish Information Processing Society, Lublin.

*Rychlicki-Kicior, K.* (2010). Java EE 6. Programming web applications. Helion, Poland (in Polish).

Стаття надійшла до редакції 30.11.2014.