

AN ANALISYS OF RDBMS, OOP AND NOSQL CONCCEPTS

*Dragos-Paul POP**¹
*Edward Cristi BUTOIU*²

ABSTRACT

Software applications often work with data. This data needs to be managed in careful way as to not be easy corrupted or lost. At the same time, programmers need a way to access this data and work with it. Classically, data was stored in a database that, more often than not, used the well known relational model. Programmers access these systems directly, trough and API, or they use Object Relational Mapping systems, a relatively old term (the first ORM, for SmallTalk, TopLink was released way back in 1994). A more recent database technology term is NoSQL (reinvented in 2009). We describe this term and what the systems that implement the concept do in the second part of the paper.

Keywords: SQL, RDBMS, ORM, NoSQL, MapReduce, OOP

1. INTRODUCTION

In the early 1960's, the database term was introduced as a simple support for what will represent the structure of a system. Besides that the splitting of data applications was a new and viable concept, it has opened the way to more robust-like applications. In that moment the databases consisted of devices that were using tape, but immediately they became easy to use, being used on systems with direct disc-access.

In 1970, Edgar Codd has proposed a more efficient data-storage method – the relational model. This model was also using SQL to allow applications to find table stored data. This relational model was almost identical with what we call today the traditional relational model. Although this model was generally accepted, until the 1980s there weren't hardware devices which could fully use this model's advantages. Until 1990, the hardware industry managed to evolve at a more adequate level, thus the relational model became a dominant method for storing data.

As well as any other technology domain, the relational database management systems competition started. Some examples of RDBMs are Oracle, Microsoft's SQL Server, MySQL and PostgreSQL.

After the year 2000 applications began to output huge amounts of data because of complex processes. Social networks were invented. Companies wished to use their data more efficiently. The change brought database structure, performance and data-availability problems which haven't been addressed by the previous relational model. With these weak spots and the need to manage huge chunks of data more efficiently, the NoSQL term appeared.

¹ * Corresonding author. Romanian – American University. 1B, Expozitiei Blvd., district 1, code 012101, Bucharest, Romania. Email: pop.dragos.paul@profesor.rau.ro

² Romanian – American University. 1B, Expozitiei Blvd., district 1, code 012101, Bucharest, Romania. Email: edwardcristi@gmail.com

2. APPLICATION-LEVEL DATA PROCESSING OPTIMIZATION THROUGH ORM (OBJECT-RELATIONAL MAPPING) SYSTEMS

Object Relational Mapping is a programming technique used by the developers for converting data between incompatible systems. This technique is used in object oriented programming languages, thus the “object” from the ORM.

ORM systems are frequently used by developers when they interact with relational database management systems. An ORM system refers to creating classes which reflect database tables but in a business-oriented manner unlike the normalized form of a RDBMS (Relational database management system).

The need for ORM systems arose from the way data is being stored and manipulated by different DBMS's and programming languages. The two kinds of systems can be grouped as such: RDBMS's (most of them follow the relational model) and programming languages (most of them have object oriented programming features or are completely object-oriented). The methods whereby data is manipulated by the two systems are totally different.

The Relational Model

Formulated by E.F. Codd in 1969, the relational model is the main storing model used by most of the RDBMS's. The main idea of the relation model is to describe a database as collection of predicates over a finite set of variables, describing the restrictions over possible value combinations. The content of a database is a finite model (logical) anytime, for example a set of relations, one per predicate variable so all predicates are satisfied. A query for information from the database is also a predicate.

Motivation behind the relational model is to supply a declarative method for data specification and queries: the users declare what information the database has and what information they want from it and then let the RDBMS to take care of the description structures for storing data and methods of data retrieval for queries.

Normalization is another feature of relational databases. The objective of database normalization is to decompose the anomaly-ridden relations to produce more smaller and compact relations. Normalization usually includes splitting the bigger tables in smaller tables (and less redundant) and defining the relations between them. The objective is to isolate data because the additions, deletions or modifications of a field to be done in a single table and get propagated throughout the rest of the database through the defined relations.

The relational model with the normalization technique are well defined for storing data because if they are used correctly they can supply a safe, rapid and without-anomalies model of data-storing.

The SQL standard is used to query databases. It was created by IBM in 1970 to be used in System R, the first database system based on the relational model. The SQL Standard is based on algebra and relational math like the relational model therefore relational databases are using the logic for storing and manipulating data.

Object Oriented Programming

Many people learn to program for the first time using a language that is not object-oriented. Non-OOP languages can be a long list of commands. The more complex programs will group command lists in functions and subroutines, each having a special task. When the program's complexity raises allowing any function to modify any part from the data can trigger anomalies which can have cascading effect.

In contrast, the object oriented programming method encourages the programmer to store data in places not accessible directly by the rest of the program. Instead, data is being accessed through special functions, called methods, which are encapsulated with data or inherited from the object classes and behave like auxiliary elements for retrieval or modification of data. The programming construction which combines the data with a set of access and management methods of theirs is called an object.

This model is used on a larger scale because it uses structures which work with data and groups the variables and functions into object properties and methods.

Combining the relational model and object oriented programming

The two structuring models are not compatible. This is due to the fact that they are completely different. One is being used for storing and manipulating huge quantities of data and the other one is used for implementation of data processing algorithms.

A programmer uses SQL to recover and send data back to the management system of the database. Moreover, he needs to know the structure of the database, tables, relations and data-processing restrictions exactly. This is a burden for each programmer because it is still a matter that needs to be taken into account. If the structure of the database changes, a programmer needs to update his code, classes and functions.

It's clear that by issuing relational stored data in an object oriented programming language can be a difficult matter sometimes. To address this issue the ORM technique was invented. It behaves like a bridge between the two systems so that the programmer can ease his worrying related to the mode by which the data is stored and recovered and to concentrate how it's processed in the application which he creates.

ORM system functionality

This system functions like an intermediate between the programmer and a management system of a database by building queries to the database for him and getting the data back in an object-oriented manner. Normally the programmer doesn't query the database management system and doesn't need to worry about the SQL Standard because everything is being created by the ORM system. In exchange, the programmer queries the ORM system.

The ORM system is actually a library (sometimes created by the programmer) which is designed to work with one or more database management systems.

ORM is a programming technique for converting data between incompatible systems in an object oriented language. This method creates a virtual database which can be used from the programming language.

Data management tasks in object oriented programming are usually implemented by object manipulation which have often non scalar values. Yet many database management systems can store and manipulate only scalar values such as integers and strings of characters organized in tables. The programmer needs either to transform the value of the objects in simple value groups in order to be stored in the database (and then to convert them back for recovery), or to use simple scalar values in the program. ORM is used to implement the first method.

The main issue is converting the logical representations of the objects into an atomized form which can be stored into the database, while the properties of the objects and relations being preserved in order to be reloaded as objects when needed. If the storage and backup functionality is implemented, it's being said that the objects are being persistent.

Compared to other traditional exchange techniques, between an object oriented language and a relational database the ORM often reduces the volume of required code to be implemented.

The disadvantages of ORM systems occur, in general because of the high abstraction level which cloaks what actually happens in the implementation code of this concept. A variety of difficulties occur when trying to implement such a concept. These difficulties are being called the object relational impedance mismatch.

The mismatches which appear between the two systems when taking into account the object oriented programming concepts

- **Encapsulation:** object oriented programs are created by using techniques which result in encapsulated object whose representations are hidden. In an object oriented platform, the underlying properties of an object are hidden to all interfaces except the one in which is implemented alongside the object.
- **Accessibility:** In the relational thinking, the private and public concepts are thought as relative concepts rather than absolute (like in object oriented programming). The relational model and object oriented objects usually have conflicts at relativity versus the absolutism of data classification and characteristics.
- **Interface, class, inheritance and polymorphism:** under an object-oriented paradigm, the objects have interfaces which together supply the only access to the internal components of this object. On the other side the relational model uses derived relations variables (views) to supply different perspectives and restrictions in order to assure data-integrity. Similarly, the essential OOP concepts for object classes, inheritances and polymorphism are not supported by the relational database systems.

A correct link between the relational and object oriented concepts can be made if the relational database's tables are tied by the kinks found in the object oriented analysis.

The differences between the data types used by the both models should be considered.

A major asymmetry between the existent relational languages and object oriented languages lies upon the differences between the data types used by both these systems. The relational model strictly forbids reference-type attributes (pointers) while the object oriented uses this concept widely. Scalar types and operator semantics are as well very often different between these two models.

For example the majority of SQL systems support strings with different specified collations and maximum limited lengths (open text-types are usually a bottleneck for performance), while the majority of OO languages consider the alphabet as an argument for the sorting routines and the character strings which are intrinsically enlarged based on the available memory. A more subtle example would be that SQL systems usually ignore the ending white space of a character string because of comparison reasons while the OO character string libraries don't do that.

Another mismatch is related to the differences between the structural and integrity aspects of the models. In OO languages, the objects can be composed from other objects. This matter can make the conversion to a less simple relational schema difficult. This problem is due to the fact that the relational data tends to be represented in a set of global relation variables and not interspersed. Relations per se being tuple sets, all conform to the same header don't have a perfect namesake in OO languages. The restrictions in OO languages are generally not declared, but do manifest just like the logical protection around the code which operates on internal encapsulated data. The relational model on the other hand requires declarative restrictions on scalar types, attributes, relation variables and on the database as a whole.

The semantic differences are especially evident in the manipulation aspects of the context models. The relational model has a relatively small and well defined intrinsic set of primitive operators for use in queries and data manipulation while the OO languages generally manage queries and manipulations through some routines and special defined operators.

Solving the mismatch issue (impedance mismatch) for the OO programs begins with the recognizing of the differences between the utilized logical systems and minimizing, compensating for the mismatches.

There were several attempts of constructing management systems of object oriented databases (OODBMS) which avoided these mismatch issues. These had a small grade of practical success, mainly because of the OO principle limitations in data modeling. There were researches carried out about the possibility of extending the OO language capabilities by including notions such as transactional memory.

A solution which is often used for solving such a mismatch issue is to split the domain logic from the business one. In this concept, the OO language is used to solve specific relational aspects in the execution moment, avoiding if possible a static approach of data. The platforms which use this model usually have analog elements for the tuple and relation concepts. The advantages of such an approach are:

- Simple ways to build automation elements for the domain data transfer, presentation and validation;

- Lower dimension programs, fast compilation and fast loading times;
- The possibility of dynamic change of data schema;
- Restrictions check;
- Reduced complexity in model synchronization.

The raise in popularity of XML databases led to the occurrence of other alternative architectures which try to solve the mismatch between the two models. These architectures use XML technology inside the client application and native XML databases on the server by using the XQuery language for querying the data. This matter allows the usage of a single data model and of a single querying language of data (XPath) in the client application and on the persistence server.

The mismatch between the two systems when it comes to application interfaces is even more visible when it's required to build interfaces which would allow the non-technical users to manipulate data which is being stored into the database. Building such interfaces requires thorough knowledge related to the nature of different database attributes from the database (more than the nature of the attribute and data type). Usually building visual interfaces which prevents illegal transactions (transactions that would break the database restrictions) by the users is considered a best practice. This matter requires mostly duplication of the present domain logic of the database in the visual interface.

Because of a limited set of data-types the SQL standard makes the proper projection of objects and business domain a difficult matter. Also the SQL standard represents an inefficient interface between a database and an application (built in an object oriented manner or not). Anyways SQL is momentarily the only general standard accepted for querying databases. Using proprietary database querying languages is seen as a bad practice and is discouraged. Other querying languages were proposed for standardization as Business Systems 12 and Tutorial D, but neither of these were adopted on a wide scale by no RDBMS vendor.

In the actual popular RDBMS systems such as Oracle and Microsoft SQL Server, the upper problems were partially resolved. Within these products, the base functionalities of the querying language can be extended with stored programs (functions and stored procedures) written in modern OO programming languages (Java for Oracle and .Net languages for Microsoft), and these programs can be invoked in querying SQL in a transparent mode. This matter means that the user doesn't know and doesn't need to know that these routines weren't initially a part of the RDBMS in cause. The modern projection models are fully supported, thus a developer can create a library of re-usable routines in the context of multiple relational schemas.

These vendors decided to add these functionalities in the RDBMS products because they realized that despite the attempts made by the ISO Standardization Committee SQL-99 to introduce procedural functionalities in the standard SQL, this will never have the rich library set and data structures found in the actual programming languages and that application developers wait to access. Because of this the difference between the application development and database administration is at the moment undefined: the implementation of robust functionalities such as restrictions and triggers require the work

of a developer with both database administration and object oriented programming capabilities. This matter also has an effect over the split of responsibilities.

Yet, many voices think that the SQL Standard does exactly what it was designed to do, namely to facilitate querying, sorting, filtering and storing of very big sets of data. Adding object oriented capabilities to this standard would only promote a wrong architectural style by including business logic at the level of data storage, which is contrary to the initial principles of RDBMSs.

In a RDBMS the canonic form of data is represented and stored. The model of databases presumes that the RDBMS is the only responsible authority with the modifications which occur at the data level. Any other systems which facilitate replication of those modifications use only copies of the data and any other modifications which occur at the database data are usually the modifications which must be taken carefully into consideration. Yet, many application developers prefer to view the data representation contained by the objects that they use as the correct stance and view the RDBMS only as a deposit for the eventual data storage.

Splitting responsibilities between the application programmers and database administrators is a sensible topic. It usually happens that when there's necessary code modifications (to implement a new thing or new functionality) that these will cause according modifications of the database. In most organizations maintaining the database is the responsibility of its administrator. Because databases must be maintained all the time, many administrators refuse to apply modifications above the structure of the database which they consider superficial. This matter is to be understood because the database administrators are usually held accountable if after a modification of a database's structure led to data loss. So the majority of definition and functionality modifications of data are maintained at the application level because at this level the possibility to lose data is less probable. Still, as applications evolve, the schema of the database remains constant and the differences between the two systems increase in such a manner that it becomes very difficult to implement new functionalities without needing to modify important portions of the application.

The main differences between the relational model and the object oriented one are:

- **Declarative and imperative interfaces** – Relational thinking tends to use data as interface not behavior as an interface. Thus it has a tendency towards the projection philosophy in contrast with the behavioral tendency of the object oriented model.
- **Schema** – The objects won't need to follow a parent schema for the attributes or methods which an objects encapsulates, while the table's tuples must follow the schema of the entity. A given tuple must be part of only one entity. The closest thing in OO is the inheritance but is in general optional.
- **Access rules** – In relational databases the attributes are being accessed and modified through the predefined relational operators. In the OO model each class allows the creation of interfaces for modifying of the state and attributes.
- **Unique identifier** – Unique identifiers (keys) generally have a text representable form but objects don't require a visible unique identifier.

- **Normalizations** – The practices of relational normalizations are usually ignored by the OO design. Another perspective is that a collection of objects, interconnected with pointers is the equivalent of a network database which, in turn, can be seen as an extremely de-normalized relational database
- **Schema inheritance** – The majority of databases doesn't support the schema inheritance. Although such a trait could be added in theory to reduce the conflict with the OOP, the supporters of relational concepts don't believe in the utility of such hierarchical taxonomy because they tend to consider that the taxonomies based on sets and classification systems to be more powerful and flexible than Tree.
- **Structure and behavior** – The OO model is mainly based on the assurance that the program's structure is reasonable (easy to maintain, easy to understand, extensible, reusable, safe), while the relational systems concentrate on the behavior type (efficiency, adaptability, error-tolerance, logical integrity, etc.). Object-Oriented Methods generally presume that the main user of the object oriented code and its interfaces are the application developers. In the relational systems, the behavior of the system from the user's point of view is considered more important. With all these, the relational queries and views are common techniques to reproduce information in specific applications configurations.
- **Multiple relations and Graph** – The relation between different elements (objects or records) tend to be treated differently than the two models. Relational relations are usually based off expressions taken from the theory of sets, while the relations between objects are based off expressions adopted from the theory of graphs (including Tree). Although each model can represent the same information like the other, the approach which they offer for the access and management of information differs.

3. APPLICATION-LEVEL DATA MANAGEMENT USING NOSQL

The NoSQL term isn't an abbreviation for "no SQL", in fact it's an acronym for "not only SQL". NoSQL databases are a group of long-lasting solutions which aren't using the relational model and they don't use SQL for queries. In addition, NoSQL wasn't launched in order to replace the relational systems but for being a support for the lack of features relational databases have.

The first special aspect of NoSQL databases is the data structure. There is a variety of methods in which NoSQL databases can classify themselves.

Classifying NoSQL databases

NoSQL databases (most of them) can get classified in 4 major categories:

- **Key-Value stores:** Saving data with a unique key and a value. Their simplicity allows them to be extremely fast and adaptable to big quantities of data
- **Column stores:** They're similar to the relational database ones but instead of records they store all the associated column values into an array.

- Document stores: Saving data without being integrated into a schema, grouping a multitude of key-value pair in a single object. This structure is similar to an associative table.
- Graph databases: Data is being stored in a flexible graph which contains a node for each object. The nodes have properties and relations with other nodes.

In principle, NoSQL databases are easily scalable because they are based off distributed systems and ignores the ACID model.

When there's a discussion about a distributed system (not necessarily database related), there's a concept that defines its limits. This is known as the CAP theorem.

The CAP Theorem

Eric Brewer introduced the CAP theorem in 2000. It dictates that in any distributed environment there's impossible to guarantee thee things at the same time, more specifically:

- **Consistency:** All the system's servers will have the same data. Therefore anyone using the system will get the most recent version of the data, no matter from what node the distributed system access started.
- **Availability:** All the servers will return data, no matter what moment.
- **Partitioning availability:** The system continues to work as a whole even if a server is offline or cannot be found in the network.

When a system becomes so pervasive that one node cannot cope with data traffic, adding additional servers for the system is usually a common solution. When adding nodes there must be a method for data splitting between these nodes. Does other several databases exists with the same data? Are there ways to place sets of data on different servers? Are other servers allowed to write data and other to read it?

No matter what option the developer would choose, the major problem lies at the synchronization of those servers. If some information is written in a specific node, how can there be any assurance that a query directed through another server will return the most recent data instead of the old ones? These events can succeed in few milliseconds. Even with a modest data collection, this matter can be extremely complex.

When it's absolutely necessary that all clients to have access to a valid database image, a node's users will have to wait for all the nodes to get to an accord before reading or writing into the database.

In this case we can observe how the availability loses priority over data integrity. In any case, there are situations when availability becomes a priority: any system node should take decisions based on its own status. If the situation requires that the system to operate on intensive-like traffic conditions and nodes have to get to a consensus for each data-accessing then the servers are in danger of failing. If the developer is interested into scalability, any algorithm which forces servers to get to an integrity consensus will lead to problems.

The Acid Model

ACID is a set of properties which get applied in transactions between databases and lies at the relational model's base. While the transactions are quite useful, they're the cause of read and write lagging in the context of them being used in relational databases. ACID is made out of four major properties:

- **Atomicity:** This is an essential concept when there is data manipulation. All transitioned data must be sent in succession or no change will be registered. This is a key property when money or considerations are interacting with the system, requiring a verification method.
- **Consistency:** Data will be accepted only if it passes all database's validations such as type, value, event restrictions.
- **Durability:** Once the data is saved, they're protected against errors, destruction or software anomalies.

Advantages of NoSQL databases

Along with NoSQL's introduction, many advantages showed up, such as:

- It allows things that weren't allowed in the past by the relational database's processing power
- Client's data is scalable and flexible, allowing a necessary scalability politic approach since the installation moment.
- There are new data types to be taken into consideration. There won't be the need for a developer to force a specific data-type which wouldn't have any logic in the context.
- Data-Writing is being made extremely fast.

As it can be seen, there are few clear advantages of NoSQL databases, but there's also negative aspects.

Disadvantages of NoSQL databases

Meanwhile there's also shorts which NoSQL systems have:

- There's no reference standard, each database behaves differently than the others.
- Queries aren't using the well-known SQL model to find records.
- NoSQL is a new technology and it's in a constant improvement.
- There are newer non-standard models with operable data, this matter sometimes can be ambiguous for developers when it comes to data classifying
- Because NoSQL avoids the ACID model, there's no guarantee that the writing of data will be successfully done.

Using NoSQL systems

After presenting the advantages and disadvantages, we are presenting the situations where NoSQL can be used as an efficient management instrument:

- Applications who write lots of data

- Applications in which the schema and structure can get modified any time
- Huge chunks of unstructured or semi-structured data
- Relational database are quite restrictive for the developer and he'll need something new.

Situations in which NoSQL databases should be avoided are the following:

- Any application that implies having money or value exchange transactions. The result would be catastrophic, knowing that NoSQL is avoiding the ACID model and data not being 100% available because of the distributed system.
- Essential or classified data of business application where the lack of a single record could cause major issues.
- Data being strongly relational between themselves, requiring the functionality of a relational system.

For all these situations, the developer should choose for a relational database which would guarantee him that data will be manipulated safely. Of course, NoSQL can be included there where this kind of manipulation has logic.

The Map/reduce concept

Map/Reduce is a software concept and it became increasingly popular in distributed processing domain. Its main concept is based off two functions – indexing and compression, both being used with a batch of input values.

The indexing function produces a result for each article in the list while the compressing function produces a single result for the whole list. CouchDB exploits the features of these two functions to produce successive calculated reports. This can be explained by the fact that each time a document is updated in the database, CouchDB, only the documents that must be modified will be reprocessed by the indexing and compression functions.

Google is using a map/reduce implementation in its web index. Google has thousands of servers which are processing terabytes of data, centralized all over the World Wide Web. Issues that can be solved by a single server within months are processed within hours because of the distribution model. The Map/Reduce library used by Google includes facilities such as traffic counterbalancing and disc-writing data optimizations, important ways of increasing the efficiency of the system. Also, the architecture is robust, thus errors or hardware malfunctions won't negatively influence the solving of the initial issue. Actually, according to public made researches on Google based on Map/Reduce, Google lost at a time 1600 from its 1800 servers from a cluster, yet the system managed to produce a result of the problem.

4. CONCLUSION

As we can see, the Realtional Model is here to stay. NoSQL systems offer great performance, but they can not completely replace RDBMSs. The best thing tot do is use what's best from both worlds, all in conjunction with an ORM system. NoSQL systems are great for storing cached objects or SQL query results, for example. Because of their

speed, they are a great source for primary data for applications that need data fast but don't need it to be consistent. For consistency we can fall back to an RDBMS.

5. ACKNOWLEDGEMENTS

The article has been supported by scientific research within the project entitled "**PRACTICAL SCHOOL: Innovation in Higher Education and Success on the Labour Market**", project identified as POSDRU/156/1.2/G/132920. The project is co-financed by the European Social Fund through the Sectorial Operational Programme for Human Resources Development 2007-2013. Investing in people!

REFERENCES

- Abramova, V., & Bernardino, J. (2013). NoSQL databases. In Proceedings of the International C* Conference on Computer Science and Software Engineering - C3S2E '13 (pp. 14–22). ACM Press. <http://doi.org/10.1145/2494444.2494447>
 - Cattell, R. (2011). Scalable SQL and NoSQL data stores. ACM SIGMOD Record.
 - Halpin, T. (1998). Data modeling in ORM. In Handbook on Architectures of Information Systems (pp. 81 – 102). Springer-Verlag. http://doi.org/10.1007/3-540-26661-5_4
 - Halpin, T. (2000). Entity Relationship Modeling from an ORM Perspective: Part 1. Journal of Conceptual Modelling (2000: 12), (December), 1–10. Retrieved from <http://www.orm.net/pdf/JCM11.pdf>
 - Han, J., Haihong, E., Le, G., & Du, J. (2011). Survey on NoSQL database. In Proceedings - 2011 6th International Conference on Pervasive Computing and Applications, ICPCA 2011 (pp. 363–366).
 - Hecht, R., & Jablonski, S. (2011). NoSQL evaluation: A use case oriented survey. In Proceedings - 2011 International Conference on Cloud and Service Computing, CSC 2011 (pp. 336–341).
 - Leavitt, N. (2010). Will NoSQL Databases Live Up to Their Promise? Computer.
 - Pokorny, J. (2011). NoSQL databases: a step to database scalability in web environment. In Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services - iiWAS '11 (p. 278). ACM Press. Retrieved from <http://dl.acm.org/citation.cfm?id=2095536.2095583>
 - Richardson, C. (2009). ORM in dynamic languages. Communications of the ACM.
 - Song, H., & Gao, L. (2012). Use ORM middleware realize heterogeneous database connectivity. In 2012 Spring World Congress on Engineering and Technology, SCET 2012 - Proceedings.
 - Stonebraker, M. (2010). SQL databases v. NoSQL databases. Communications of the ACM.
 - Van Zyl, P., Kourie, D. G., & Boake, A. (2006). Comparing the performance of object databases and ORM tools. Proceedings of the 2006 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries - SAICSIT '06, 1–11. Retrieved from <http://portal.acm.org/citation.cfm?doid=1216262.1216263>
- Xia, C., Yu, G., & Tang, M. (2009). Efficient implement of ORM (Object/Relational Mapping) use in J2EE framework: Hibernate. In Proceedings - 2009 International Conference on Computational Intelligence and Software Engineering, CiSE 2009.