A Comparison between a Relational Database and a Graph Database in the context of a Personalized Cancer Treatment Application

Alexandra Martinez[1], Rodrigo Mora[2], Daniel Alvarado[3], Gustavo López[3], and Steve Quirós[2]

[1] Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica, San José, Costa Rica
[2] Centro de Investigación en Enfermedades Tropicales, Facultad de Microbiología, Universidad de Costa Rica, San José, Costa Rica
[3] Centro de Investigaciones en Tecnologías de la Información y Comunicación, Universidad de Costa Rica, San José, Costa Rica
{alexandra.martinez, rodrigo.morarodriguez, daniel.alvarado_g, gustavo.lopez_h, steve.quiros}@ucr.ac.cr

Abstract. This paper presents a performance comparison between a relational database (implemented in MySQL) and a graph database (implemented in Neo4j). Unlike traditional benchmarks, this comparison is made in the context of a real health application which was developed in Costa Rica. The comparison encompassed twelve queries and three data size configurations.The results of the comparison indicate that MySQL performs better than Neo4j in most cases, but has a poor performance when data size is large and the queries have multiple join operations.

Keywords: relational databases; graph databases; cancer treatment.

1 Introduction

Information technology has been an enormous aid in the fields of medicine and healthcare in general, as it facilitates the management and sharing of very large amounts of data and its standardization. We have created a software healthcare platform that supports the personalized cancer treatment process at various stages. This software platform will be used by oncologists, pharmacists and pathologists from the hospitals as well as by microbiologists and technicians from the cancer lab. The actual personalized cancer treatment that is implemented is the ATP Tumor Chemosensitivity Assay [2], which will be offered by the university cancer lab to national public hospitals.

When developing the software platform, we wondered which paradigm was better suited for the task at hand: a traditional relational model or a modern NoSQL model. As the literature review did not find studies that were similar to what we needed, we decided to build a flexible enough platform that allows us to change the underlying database transparently, so that we could experiment

with both types of database in order to decide which one worked best in our context.

The rest of the paper is organized as follows. Section 2 presents the background on personalized cancer treatment and relevant database paradigms. Section 3 describes the context. Section 4 specifies the methodology. Section 5 shows the results and Section 6 states our conclusions.

2 Background

2.1 Personalized Cancer Treatment

Cancer therapeutics are limited by the tumor's resistance to chemotherapy, which is the major obstacle for effective patient treatment [4]. In traditional clinical practice, resistance is detected during treatment [1]. However, once a tumor is resistant to a chemotherapy, it often becomes resistant to multiple chemotherapeutic drugs. Hence, treating a cancer patient with a suboptimal drug as a first-line therapy lowers her chances of survival. Ideally, the first-line treatment should be the chemotherapeutic drug that has the maximum probability of reducing tumor robustness.

Early diagnosis of resistance is achieved by combining clinical, pathological and molecular markers, and complementary in-vitro chemosensitivity tests. In-vitro chemosensitivity tests can predict which chemotherapeutic drug a patient will benefit the most (or the least) from. One such in vitro test is the ATP Tumor Chemosensitivity Assay (ATP-TCA) [2]. ATP-TCA has a predictive value of 93% for sensitivity (i.e., tumor's positive clinical response) and close to 100% for resistance (i.e., tumor's clinical nonresponse). ATP-TCA has been reported to increase tumor response rates and prolong survival times, and is particularly useful when there are many alternative treatment protocols. It may be considered the assay with best documented and validated technology [2].

In brief, early diagnosis of resistance (including in-vitro chemosensitivity tests like ATP-TCA) enables optimized and personalized cancer treatment [4, 5].

2.2 Database Paradigms

The Relational Paradigm The relational model, proposed by Edgar Codd in 1970, has been the predominant paradigm. However, it is currently facing strong competition from modern alternatives like object-relational and NoSQL paradigms. The relational model represents data as a collection of relations, and a relation is essentially a table of values where each row represents a set of related values Column headers represent the attributes we want to store and every row shares these attributes, but not their values. Relational databases work best with structured data, which fit easily into tables [3]. Formally, operations on this model can be perfomed through relational algebra or relational calculus, but the most popular query language is SQL (Structured Query Language), an ANSI (American National Standards Institute) standard. Typically, database

applications use transactions to encapsulate a series of operations into a unit of work Transactions have four main properties (known as ACID): Atomicity, Consistency, Isolation and Durability, which together guarantee the reliability of a relational database [8].

The Graph Paradigm Graph oriented database management systems are designed to facilitate relationships between the nodes. Instead of using foreign keys to represent a relationship, graph databases use arcs that directly connect two nodes. Operations on this model can be performed through a graph query language.

Graph databases are one of the four categories of NoSQL databases. The other categories of NoSQL databases are: key-value, documents and column oriented. NoSQL database management systems emerged as a response to the limitations of the relational technology and the demands of the Web 2.0 age. They were born with the advent of MapReduce and BigTable in 2004 and 2006, respectively [7]. The NoSQL movement has since grown rapidly, to the point that today there are more than 225 NoSQL storage systems (according to http://nosql-database.org, retrieved on February 24th, 2016). The term "NoSQL" (which the community has interpreted as *not only SQL*) denotes the next generation of database management systems, that are mostly non-relational, distributed, open source and horizontally scalable. Additionally, they are often schema-free, support easy replication, have a simple API, are eventually consistent (BASE instead of ACID) and can handle huge amount sof data. NoSQL database management systems generally process data quicker than relational ones, partly because of their simpler data models and because they don't have to commit to certain restrictions imposed by the ACID properties [3].

NoSQL paradigm may never completely replace relational paradigm, but it might become a better option for projects that work with unstructured data and require scalability [3].

3 Context

The databases used in this comparison were developed as a part of a software platform that supports personalized cancer therapy based on the ATP-TCA assay. The purpose of this platform is to facilitate the collection, storage, management, and communication of ATP-TCA assays data. A detailed description of the design of this platform was described in [6]. All the queries used in the comparison correspond to real operation requirements of the platform, although not all of them are currently accesible from the web application.

The architecture of this platform consisted of three layers: a front-end, a back-end, and a web services layer that allows communication between the other two. The front-end was a web application developed with Microsoft .NET. The back-end consisted of either a relational database that was implemented in MySQL, or a graph database that was implemented in Neo4J. Web services were used to

allow for transparent switching of back-end implementation . Given that medical information of patients is considered sensitive data, we implemented access control through the well established and accepted ASP.NET security, authentication and authorization framework. Access control was built within the front-end application since the server that runs it is the only one that can be publicly accessed; all other servers are configured in a private and protected network.

4 Methodology

An experiment was designed to compare the performance of the relational and graph databases. The performance metric used was the mean response time of each database engine to a specific set of queries. The same set of queries was executed against each database engine (MySQL and Neo4j). To make sure that the queries performed on both engines were equivalent, for each pair we compared and verified that the returned result was the same. (We needed to do this since the formulation and syntax of the same query was drastically different in each database).

4.1 Experimental configuration

We used virtual machines mounted on a virtualization server with eight 2.93 GHz Intel Xeon processors, 32GB of RAM memory and 8TB of secondary storage. This virtualization server ran an ESXi-5.5.0-1331820-standard. Each of the virtualized machines used Debian 7 (wheezy). We used version 14.14 Distrib 5.5.38 of MySQL for Debian and version 1.8.1 of Neo4J.

When installing and configuring the Neo4j and MySQL database management systems (each on one server), we followed the instructions published in their respective websites, to avoid favoring one of them with special tunings. After these servers (virtual machines) were configured, we cloned them, following the instructions provided by VMware ESXi. A *clean* version of each virtual machine was stored before loading the data. We did not consider different alternatives of indexes, blocking schemes or hashing structures; instead we used the default ones that come with each database management system.

The experiments were performed using three different datasets of increasing size: the first dataset had 1.000 entries per table or node type, the second dataset had 10.000 entries per table or node type, and the third dataset had 100.000 entries per table or node type. Each dataset was generated with *generatedata.com*, a free and open source tool created by Benjamin Keen. All data was randomly generated, except for the primary keys and the foreign keys (referenced attributes in the graph database), which were generated in ascending order so that data could be related in a coherent and automated way. Automatic refencing of primary keys (attributes in nodes) from foreign keys was possible through a script written by us.

The deployed database servers were not optimized for their virtual machine's RAM size or other characteristics. All the virtual machines had the same characteristics: 2048 MB of RAM, 32-bit operating systems, 30.93 MB of memory

overhead, 16.11 GB of provisioned storage, and one processor (2.93 GHz Intel Xeon processor). To assure the estability of the virtualization server while running the tests, only one virtual machine was running at a time. Moreover, the virtual machines hosting the database servers did not have internet connection, and a direct ethernet connection was used to send the quieries and gather the results.

We chose JMeter as the tool for executing the tests. This tool allowed us to establish similar execution parameters for the two databases (for example, flushing the database cache before each query). We configured it to execute five threads, each running a loop of 50 calls, which resulted in 250 executions of each query.

4.2 MySQL Database

The relational data schema had 28 tables (relations) in total. The conceptual schema of the implemented database is shown in Fig. 1. No index was added to the basic implementation of this database schema, because we intended to evaluate its performance while trying to avoid any bias due to its design or implementation.

4.3 Neo4j Database

The graph database had 22 node types and 23 relationship types. We talk about node types and relationship types due to the data organization nature, where there is no predefined schema equivalent to relational tables. Each node may have different characteristics without having to comply with established rules or design restrictions. Also, each relationship instance depends directly on the nodes it connects. The logical schema of the database implemented is shown in Fig. 2. Likewise, no index was added to the basic implementation of this database schema.

4.4 Queries

The system was evaluated using twelve queries, which are described next. In MySQL, the queries were implemented with SQL while in Neo4j, the queries were implemented with Cypher (a declarative graph query language, inspired on SQL).

Query 1 Returns the doctors who have participated in a patient's treatment. In SQL, this query includes two join operations over three tables (see code below), while in Cypher, it uses three node types and two relationship types.

```
SELECT p.Name, p.Id, d.Name, d.Id
FROM Doctor d INNER JOIN FollowUp f ON f.DoctorId = d.Id
    INNER JOIN Patient p ON p.Id = f.PatientId
WHERE p.Name = 'Gwendolyn'
ORDER BY p.Id
```

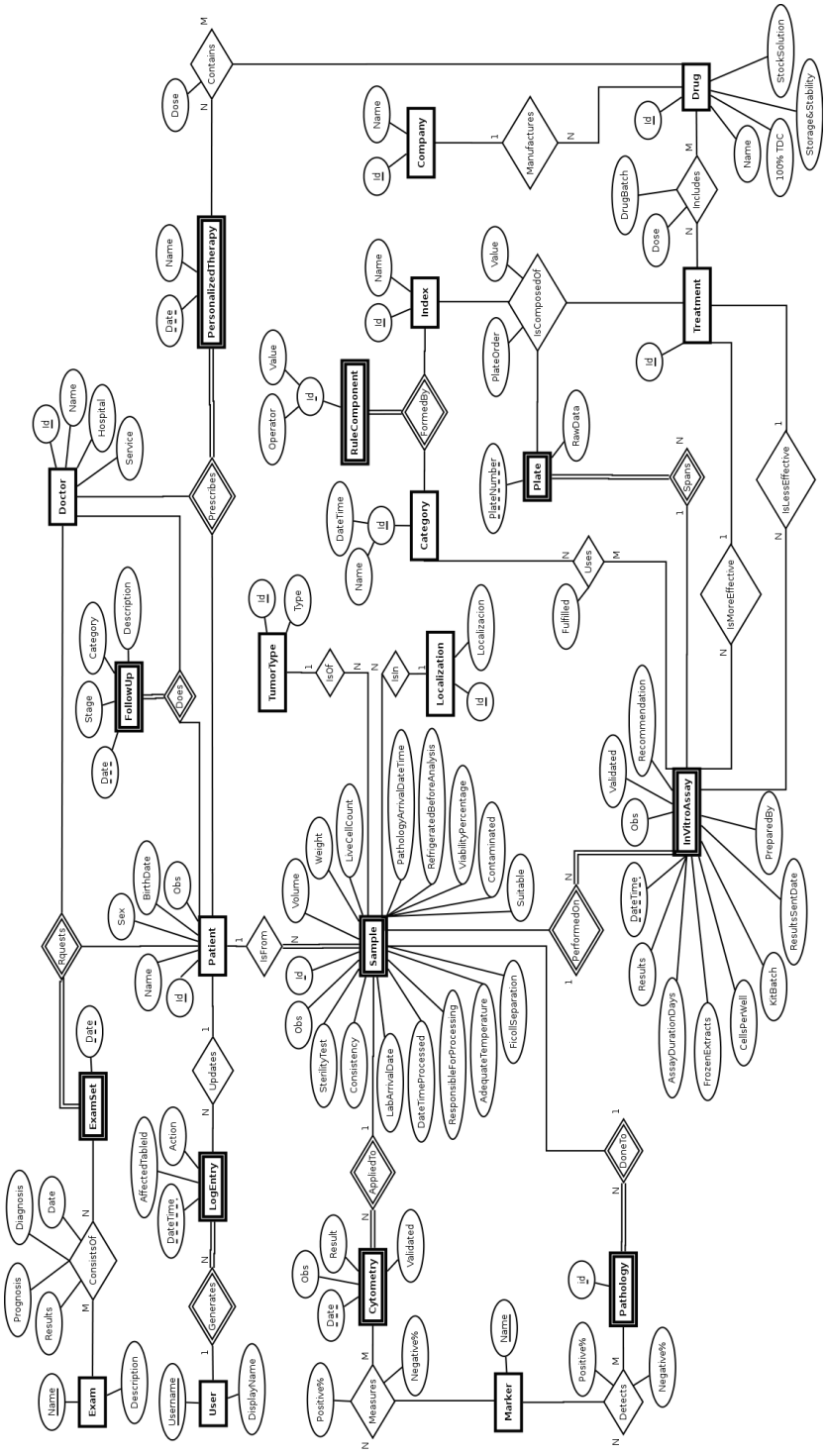

Fig. 1. Conceptual schema of the relational database.

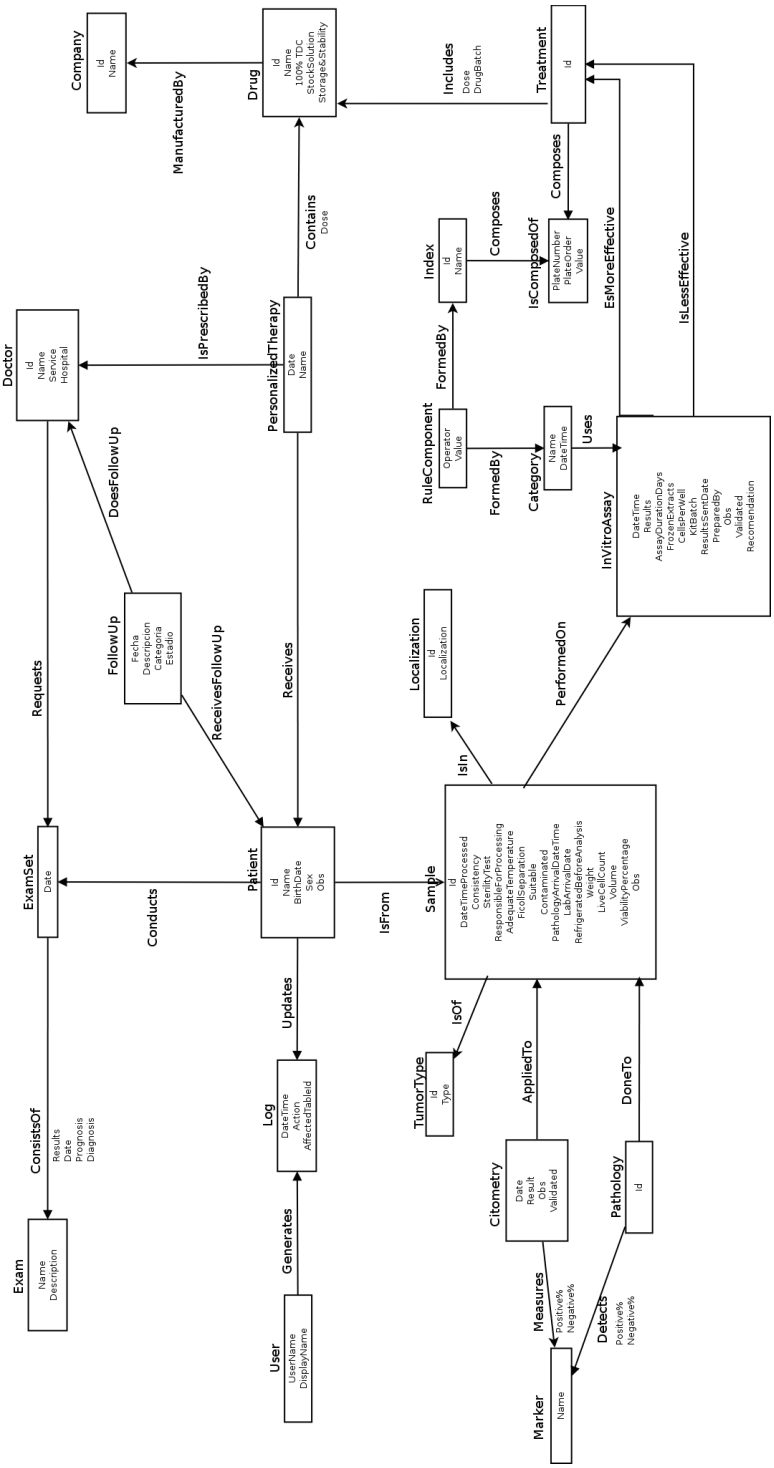

Fig. 2. Logical schema of the graph database.

Query 2 Returns the name, date and result of all medical exams a patient has undergone. In SQL, this query includes just one table (see code below), while in Cypher, it uses two relationship types.

```
SELECT Name, Date, Results
FROM ConsistsOf
WHERE PatientId = 200001094
ORDER BY Date DESC
```

Query 3 Retrieves the number of samples for each tumor type. In SQL, this query includes one join operation between two tables (see code below), whereas in the graph schema it includes one relationship type.

```
SELECT Name, count(*)
FROM Sample s INNER JOIN TumorType t ON t.Id = s.Id
GROUP BY Name
ORDER BY Name
```

Query 4 Returns the average, minimum and maximum age across all patients. The SQL code for this query is:

```
SELECT avg(TIMESTAMPDIFF(YEAR, BirthDate, CURDATE())),
    min(TIMESTAMPDIFF(YEAR, BirthDate, CURDATE())),
    max(TIMESTAMPDIFF(YEAR, BirthDate, CURDATE()))
FROM Patient
```

Query 5 Retrieves the percentage of inadequate samples. The SQL code is:

```
SELECT ((select count(*) FROM Sample WHERE suitable = 0)
/ (select count(*) FROM Sample))
from Sample LIMIT 1
```

Query 6 Obtains the average, minimum and maximum values for each index (like IC90, IC50, AUC, etc.) across all chemosensitivity assays. In SQL, this query includes one join operation between two tables (see code below), whereas in Cyhper it includes one relationship type.

```
SELECT nombre, avg(Value), min(Value), max(Value)
FROM FormedBy INNER JOIN Index on Id = IndexId
GROUP BY Name ORDER BY Name
```

Query 7 Retrieves a patient's information based on her ID. In SQL, this query involves just one table (see code below), and in Cypher, it involves one node type.

```
SELECT * FROM Patient
WHERE Id = '200001002'
```

Query 8 Retrieves a patient's information based on her name. In SQL, this query involves a single table (see code below), while in Cypher, it involves one node type.

```
SELECT * FROM Patient
WHERE Name = 'Gwendolyn'
```

Query 9 Returns the pathology and cytometry markers obtained for each female patient. In SQL, this query involves three join operations over four tables (see code below), while in Cypher, it involves five relationship types.

```
SELECT p.Id, m.Name, d.Name
FROM Patient p JOIN Detects d ON p.Id = d.PatientId JOIN
    Measures m ON p.Id = m.PatientId JOIN Cytometry c ON m
    .Date = c.Date AND m.SampleId = c.SampleId
WHERE p.Sex = true AND c.Validated = true
ORDER BY p.Id
```

Query 10 Retrieves the drugs (and the companies that produces them) used in each chemosensitivy assay. In SQL, this query involves three join operations over four tables (see code below), while in Cypher, it involves four relationship types.

```
SELECT DISTINCT c.DateTime, d.Name, y.Name
FROM Drug d JOIN Includes t ON d.Id = t.DrugId
JOIN IsComposedOf c ON c.t.TreatmentId = t.TreatmentId
    JOIN Company y ON d.Id = y.IdDrug
ORDER BY c.DateTime
```

Query 11 Retrieves all adequate samples. In SQL, this query involves just one table (see code below), while in Cypher, it involves one node type.

```
SELECT * FROM Sample
WHERE Suitable <> 0
```

Query 12 Retrieves all samples of a patient, based on her ID. In SQL, this query involves just one table (see code below), and in Cypher, it involves one node type.

```
SELECT * FROM Sample
WHERE PatientId = 200001053
```

5 Results

Next we present the results obtained from our experiment. Fig. 3 shows the mean response time in miliseconds (ms) of MySQL and Neo4J for each of the

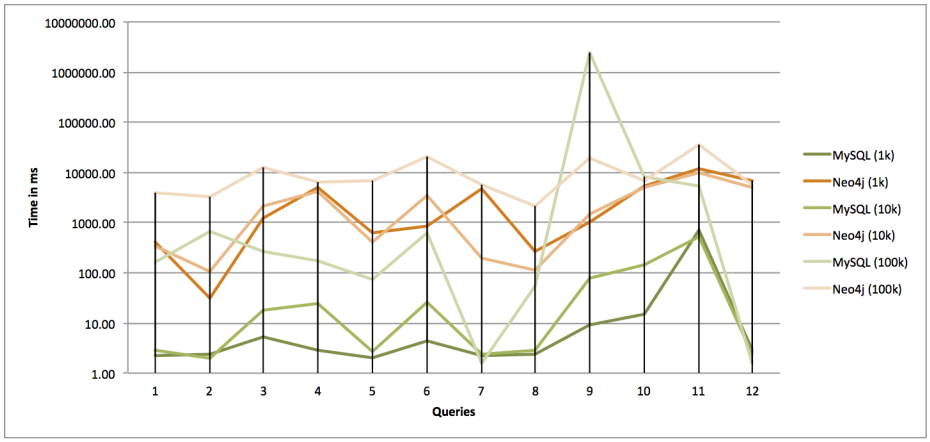

Fig. 3. Query performance of MySQL and Neo4j for datasets of 1k, 10k and 100k sizes.

12 queries, under the three datasets used. From Fig. 3 we observe that, for most queries, MySQL performs better than Neo4j (green lines are below orange lines most of the time). Furthermore, for some queries Neo4j shows worse performance than MySQL even when comparing different data sizes: 1k for Neo4j and 100k for MySQL (i.e., MySQL sometimes outperforms Neo4j even if the size of the data is 3 orders of magnitude larger). However, for query 9, the performance of MySQL is much worse than Neo4j at 100k data size (surpassing Neo4j by more than 2 orders of magnitude). Query 9 is probably the most complex query in terms of data that needs to be related (join operations in the relational model), since it requires three joins over four tables plus an ordering operation (order by clause), so it seems data size takes a toll on performance when queries are complex. It is well known that joins are costly operations in relational databases, hence we expect to see a performance degradation as data (table) size increases in the presence of multiple joins. Likewise, for query 10 the performance of MySQL is worse than Neo4j at 100k data size, and this query also has 3 joins plus an ordering operation.

6 Conclusion

We compared the performance of a relational database (implemented in MySQL) and a graph database (implemented in Neo4j), in the context of a health application for personalized cancer treatment in Costa Rica. A detailed description of the methodology was offered, including the experimental setup. The comparison encompassed twelve queries and three data size configurations: all tables or node types with 1.000 entries, 10.000 entries and 100.000 entries. The results of the experiment indicate that MySQL performs better than Neo4j in most cases, but Neo4j outperforms MySQL in two queries that require multiple join operations when data size is 100.000 entries per table or node type.

Acknowledgments

This work was partially supported by Research Center for Communication and Information Technologies (CITIC) and by Computer Science and Information Department (ECCI), at the University of Costa Rica.

References

1. Kitano, H.: Cancer as a robust system: implications for anticancer therapy. Nat Rev Cancer 4(3), 227–235 (03 2004)
2. Kurbacher, C.M., Cree, I.A.: Chemosensitivity testing using microplate adenosine triphosphate-based luminescence measurements. Methods Mol Med 110, 101–120 (2005)
3. Leavitt, N.: Will nosql databases live up to their promise? In: Computer. vol. 43, p. 1214. IEEE Computer Society (2010)
4. Lippert, T.H., Ruoff, H.J., Volm, M.: Intrinsic and acquired drug resistance in malignant tumors. the main reason for therapeutic failure. Arzneimittelforschung 58(6), 261–264 (2008)
5. Lippert, T.H., Ruoff, H.J., Volm, M.: Intrinsic and acquired drug resistance in malignant tumors. the main reason for therapeutic failure. Arzneimittelforschung 58(6), 261–264 (2008)
6. Martinez, A., Mora, R., Lpez, G., Bolaos, C., Alvarado, D., Solano, A., Lpez, M., Quirs, S., Bez, A.: New Advances in Information Systems and Technologies, chap. Design and Evaluation of a Personalized Cancer Treatment System using Human-Computer Interaction Techniques. Springer International Publishing (2016)
7. Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D.: Database research: Are we at a crossroad? reflection on nosql. In: Proceedings of the 48th Annual Southeast Regional Conference. ACM (2010)
8. Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D.: A comparison of a graph database and a relational database. In: Proceedings of the 15th International Conference on Network-Based Information Systems. IEEE (2012)