NOSOLAP: Moving from Data Warehouse Requirements to NoSQL Databases

Deepika Prakash
Department of Computer Engineering, NIIT University, Neemrana, Rajasthan, India

Keywords: Data Warehouse, NoSQL Databases, Star Schema, Early Information Model, ROLAP, MOLAP, NOSOLAP.

Abstract: Typical data warehouse systems are implemented either on a relational database or on a multi-dimensional database. While the former supports ROLAP operations the latter supports MOLAP. We explore a third alternative, that is, to implement a data warehouse on a NoSQL database. For this, we propose rules that help us move from information obtained from data warehouse requirements engineering stage to the logical model of NoSQL databases, giving rise to NOSOLAP (NOSql OLAP). We show the advantages of NOSOLAP over ROLAP and MOLAP. We illustrate our NOSOLAP approach by converting to the logical model of Cassandra and give an example.

1 INTRODUCTION

Traditionally, Data Warehouse (DW) star schemas are implemented either using a relational database which allows ROLAP operations or on a multi-dimensional database that allows MOLAP operations. While the data in the former is stored in relational tables, the data in multidimensional databases (MDB) are either in the form of multi-dimensional array or hypercubes. A number of RDBMS offer support for building DW systems and for ROLAP queries. MOLAP engines have proprietary architectures. This results in niche servers and is often a disadvantage.

Another emerging approach is to use NoSQL databases for a DW system. Data of a NoSQL database is not modelled as tables of a relational database and thus, NoSQL systems provide a storage and retrieval mechanism which is different from relational systems. The data models used are key-value, column, document, and graph.

The motivation of using a NoSQL database lies in overcoming the disadvantages of relational database implementations. These are:

i. Today, there is a need to store and process large amounts of data which the relational databases may find difficult (Chevalier et al., 2015; Stonebraker, 2012). Further, relational databases have difficulties in operating in a distributed environment. However, there is a need to deploy DWs on the cloud (Dehdouh et al., 2015), in a distributed environment (Duda, 2012). A relational database does not provide these features while guaranteeing high performance.

ii. It may be the case that some piece of data is not present in underlying data sources at the time of extraction (ETL). In a relational database engine this is handled by using a NULL 'value'. This causes major difficulties particularly in the use NULL as a dimension value and also as a foreign key value in fact tables. Rather than use NULL values, star schema designers use special values like -1, 0, or 'n/a' in dimensions. It may also be required to use multiple values like 'Unknown' and 'Invalid' to distinguish between the different meanings of NULL. For facts that have NULL values in their foreign keys, introduction of special dimension rows in dimension tables is often proposed as a possible solution to ease the NULL problem.

Designers of star schemas have outlined a number of problems associated with these practical solutions to the problem of NULL values. Some of these are, for example, difficulty of forming queries, and misinterpretation of query results.

The problem of NULL values can be mitigated in a NoSQL database because these systems

Prakash, D.
NOSOLAP: Moving from Data Warehouse Requirements to NoSQL Databases.
DOI: 10.5220/0007748304520458
In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2019)*, pages 452-458
ISBN: 978-989-758-375-9

completely omit data that is missing. Thus, the problem of NULLS in the data cube is removed.

iii. DW 2.0 (Inmon, 2010) states that DW is to cater to storing text, audio, video, image and unstructured data "as an essential and granular ingredient". A relational database fails when it comes to unstructured, video, audio files. A NoSQL database can provide a solution to this disadvantage.

iv. ETL for a relational implementation of a DW is a slow, time consuming process. This is because data from disparate sources must be converted into one standard structured format of the fact and dimension tables. We believe that since structured data is not mandated by NoSQL databases, ETL will be faster.

v. ROLAP has heavy join operations making the performance of the system slow. We believe performance of DW systems can be improved if implemented in a NoSQL database.

vi. Relational databases focus on ACID properties while NoSQL databases focus on BASE properties. Since a DW largely caters to a read-only, analytic environment with changes restricted to ETL time, enforcement of ACID is irrelevant and the flexibility of BASE is completely acceptable and, indeed, may lead to better DW performance.

There is some amount of work that has been done in implementing the DW as XML databases and also on NoSQL platform. The basic idea is to arrive at facts and dimensions and then convert, in the latter, the resulting multi-dimensional structure into NoSQL form. By analogy with ROLAP and MOLAP we refer to this as NOSOLAP. The NoSQL databases considered so far, in NOSOLAP, are column, and document databases.

(Chevalier et al., 2015) converted the MD schema into a NoSQL column oriented model as well into a document store model. We will consider each of these separately. Regarding the former, each fact is mapped to a column family with measures as the columns in this family. Similarly each dimension is mapped to separate column families with the dimension attributes as columns in the respective column families. All families together make a table which represents one star schema. They used HBase as their column store.

The work of (Dehdouh et al., 2015) is similar to the previous work but they introduce simple and compound attributes into their logical model. (Santos et al., 2017) transforms the MD schema into HIVE tables. Here, each fact is mapped to a primary table with each dimension as a descriptive component of the primary table. The measures and all the non-key attributes of the fact table are mapped to the analytical component of the primary table. As per the query needs and the lattice of cuboids, derived tables are constructed and stored as such.

Now let us consider the conversion of (Chevalier et al., 2015) into document store. Each fact and dimension is a compound attribute with the measures and dimension attributes as simple attributes. A fact instance is a document and the measures are within this document. Mongodb is the document store used.

Since the foregoing proposals start off from a model of facts and dimensions, they suffer from limitations inherent in the former. Some of these are as follows:

i. Since aggregate functions are not modelled in a star schema, the need for the same does not get translated into the model of the NoSQL database.

ii. Features like whether history is to be recorded, what is the categorization of the information required, or whether a report is to be generated are not recorded in a star schema

Evidently, it would be a good idea to start from a model that makes no commitments to the structural, fact-dimension, aspects of a data warehouse and yet captures the various types of data warehouse contents. In going to NOSOLAP, we now state our position in this position paper: *we propose to move from a high-level, generic, information model to NoSQL databases directly, without the intervening star schema being created*. The consequence of this direct conversion is the elimination of the step of converting to a star schema.

To realize our position, we will need to reject all Data Warehouse Requirements Engineering, DWRE, techniques that produce facts and dimensions as their output. Instead, we will look for a DWRE approach that outputs data warehouse contents in a high-level information model that captures in it all the essential informational concepts that go into a data warehouse.

In the next section, section 2, we do an overview of the different DWRE approaches and identify a generic information model. This model is described in section 3. Thereafter, in section 4, we identify Cassandra as the target NoSQL database and present some rules for conversion from the generic information model to the Cassandra model. Section 5 is the concluding section. It summarizes our position and contains an indication of future work.

2 OVERVIEW OF DWRE

In the early years of data warehousing, the requirements engineering stage was de-emphasized. Indeed, both the approaches of Inmon (Inmon, 2005) and Kimball (Kimball, 2002) were for data warehouse design and, consequently, do not have an explicit requirements engineering stage. Over the years, however, several DWRE methods have been developed. Today, see Figure. 1, there are three broad strategies for DWRE, goal oriented (GORE) approaches, process oriented (PoRE) techniques and DeRE, decisional requirements engineering.

Some GORE approaches are (Prakash and Gosain, 2003; Prakash et al., 2004; Giorgini et al., 2008). In the approach of (Prakash and Gosain, 2003; Prakash et al., 2004) there are three concepts, goals, decisions and information. Each decision is associated with one or more goals that it fulfils and also associated with information relevant in taking the decision. For this, information scenarios are written. Facts and dimensions are identified from the information scenarios in a two step process.

In GrAND (Giorgini et al., 2008) Actor and Rationale diagrams are made. In the goal modelling stage, goals are identified and facts associated as the recordings that have to be made when the goal is achieved. In the decision-modeling stage, goals of the decision maker are identified and associated with facts. Thereafter dimensions are associated with facts by examining leaf goals.

PoRE approaches include Boehnlein and Ulbricht (Boehnlein and Ulbrich-vom, 1999; Boehnlein and Ulbrich-vom, 2000). Their technique is based on the Semantic Object model, SOM, framework. The process involves goal modelling, modelling the business processes that fulfil the goal. Entities of SERM are identified which is then converted to facts and dimensions; facts are determined by asking the question, how can goals be evaluated by metrics? Dimensions are identified from dependencies of the SERM schema.

Another approach is by Bonifati (Bonifati et al., 2001). They carry out goal reduction by using the Goal-Quality-Metric approach. Once goal reduction is done, abstraction sheets are built from which facts and dimensions are identified. In the BEAM* approach (Corr and Stagnitto, 2012) each business event is represented as a table. The attributes of the table are derived by using the 7W framework. 7 questions are asked and answered namely, Who is involved in the event? (2) What did they do? To what is done? (3) When did it happen? (4) Where did it take place? (5) Why did it happen? (6) HoW did it happen – in what manner? (7) HoW many or much was recorded – how can it be measured? Out of these, the first six form dimensions whereas the last one supplies facts.

Whereas both GORE and PORE follow the classical view of a data warehouse as being subject oriented, DeRE takes the decisional perspective. Since the purpose of data warehouse is to support decision-making, this approach makes decision identification the central issue in DWRE. The required information is that which is relevant to the decision at hand. Thus, DeRE builds data warehouse units that cater to specific decisions. Information elicitation is done in DeRE using a multi factor approach (Prakash, 2016; Prakash and Prakash, 2019). For each decision its Ends, Means, and Critical Success Factors are determined and these drive information elicitation

Figure 1 shows the two approaches to representation of the elicited information, the multi-dimensioned and the generic information model approaches. The former lays emphasis on arriving at the facts and dimensions of the DW to-be. The latter, on the other hand, is a high level representation of the elicited information. The dashed vertical lines in the figure show that in both GORE and PoRE the focus of the requirements engineering stage is to arrive at facts and dimensions. On the other hand, the DeRE approach represents the elicited information in a generic information model.

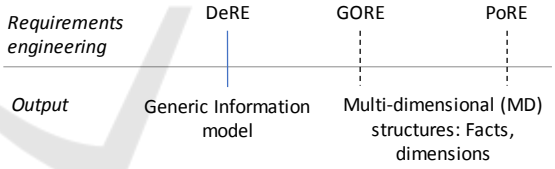

Figure 1: DWRE Techniques and their Outputs.

(Prakash and Prakash, 2019) model information as early information. Information is early in the sense that it is not yet structured into a form that can be used in the DW to-be. An instantiation of this model is the identified information contents of the DW to-be.

Since the model is generic, it should be possible to produce logical models of a range of databases from it. This is shown in Figure 2. As shown, this range consists of the multi-dimensional mode, XML schema, and NoSQL data models. Indeed, an algorithmic approach was proposed in (Prakash, 2018) to identify facts, dimensions and dimension hierarchies from the information model.

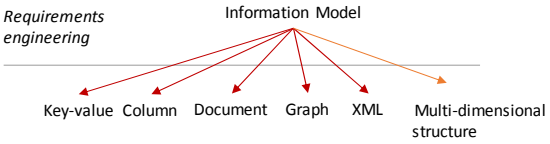

Figure 2: Converting the Information Model.

We propose to use the information model of (Prakash, 2018) to arrive at the logical schema of NoSQL databases.

3 THE INFORMATION MODEL

The information model of (Prakash and Prakash, 2019), shown in Figure 3, tries to capture all information concepts that is of interest in a data warehouse. Thus, instead of just focusing on facts and dimensions, it details the different kinds of information.

Information can either be detailed, aggregate or historical. Detailed information is information at the lowest grain. Aggregate information is obtained by applying aggregate functions to either detailed or other aggregate information. For example, items that are unshipped is detailed information whereas total revenue lost due to unshipped items is aggregate information as function sum need to be applied to the revenue lost from each unshipped item. Information can also be historical. Historical information has two components, time unit that states the frequency of capturing the information and duration which states the length of time the data needs to be kept in the data warehouse.

Information can be computed from other detailed, aggregate or historical information. Information is categorized by category which can have one or more attributes. A category can contain zero or more categories. Information takes its values from a value set.

Let us consider an example from the medical domain. The average waiting time of the patients is to be analysed. This is calculated as the time spent by the patient between registration and consultation by a doctor. Let us say that waiting time has to be analysed department wise and on a monthly basis. Historical data of 2 years is to be maintained and data must be captured daily. Instantiation of the information model gives us:

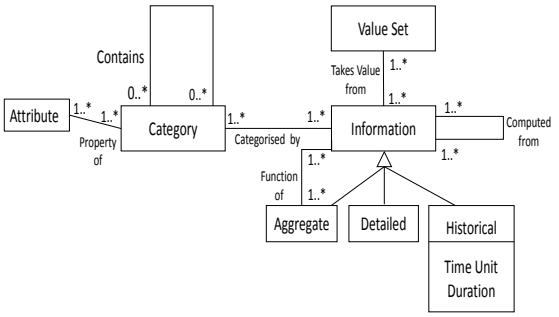

Figure 3: Information model.

Information: waiting time of patients
Computed from: registration time, consultation time
Aggregate functions: average
Category: Department wise, Month-wise
Category Attributes: Department: code, name
Month: Month
History: Time Unit: Daily
Duration: 2 years

4 MAPPING RULES

Having described the information model to be used as input to our conversion process, it is now left for us to identify the target database model and the mapping rules that produce it.

We use Cassandra as our target NoSQL database. First, since it belongs to the class of NoSQL databases, Cassandra does not suffer from the deficiencies of the relational model identified earlier. Cassandra, a blend of Dynamo and BigTable, gets its distributed feature from Dynamo. Cassandra uses a peer-to-peer architecture, not master/slave architecture and handles replication through asynchronous replication.

Cassandra is column-oriented and organizes data into tables with each table having a partitioning and clustering key. The language to query the tables is CQL which has aggregate functions like min, max, sum, average.

Cassandra has the possibility of supporting OLAP operations. In this paper, our concern is the conversion to the Cassandra logical model and we do not take a position on OLAP operations. However, we notice that the SLICE OLAP operation can be defined as a set of clustered columns in a partition that can be returned based on the query. Cassandra does indeed support the SLICE operation well. This is because it is very efficient in 'write' operations having its input/output operations as

sequential operations. This property can be used for the SLICE operation.

We give a flavour of our proposed rules that convert the information model of the previous section to the logical model of Cassandra. These are as follows:

Rule 1: Each Information, I, of the information model, becomes a table of Cassandra named I. In other words, each instantiation of the information model of figure 2 gives us one table.

Rule 2: Each 'Computed from' Ɛ I is an attribute of the Cassandra table, I.

Rule 3: Each category attribute, attr, of the information model, becomes attributes of Cassandra table, I.

Rule 4: Let us say that category C contains category c. In order to map a 'contains' relationship, two possibilities arise:

a) There is only one attribute of the category c. We propose the use of *set* to capture all the instances of c. The set, s, thus formed is an attribute of Cassandra table, I. For example, consider that a Department contains Units and Units has one attribute unit-name. Set, named say unit_name, will be an attribute of table I. Set unit_name is defined as

unit_name set <text>

where, each name is of data-type text.
The set 'unit_name' will capture all the units under the department.
Notice that since *Set*s are used to store a group of values, it is a suitable data structure for this case.

b) There is more than one attribute of the category c. Since a *Map* is used to store key-value pairs, we propose to use *Map* data structure of Cassandra. The *Map* created will form an attribute of the Cassandra table, I. Consider the same example of a Department contains Units. But this time let Units have attributes, code, name and head. A *Map* will be defined as

Units map<int, text, text>

where each code, name and head can be stored using data-type int, text and text respectively.

Rule 5: The category of the information model makes up the primary key of table I. Notice, that the model allows more than one category for information, I. Now, the primary key of Cassandra has two components namely, partitioning key and clustering key. Therefore, we need to specify which category maps to which key of Cassandra. There are two possibilities:

a) If there is only one category, then that category is the partitioning key.

b) In the second case, there may be more than one 'category'. Note that the partitioning key is used to identify and distribute data over the various nodes. The clustering key is used to cluster within a node. Thus, the decision on which category or pair of 'category' becomes the partition key and which becomes the clustering key influences the performance of the DW system. We recommend that all the categories should become partitioning keys. However, notice that after selecting a category as the partitioning key, a specific attribute of the category must be selected to designate it as a key. This task will have to be done manually.

For deciding on the clustering key, any attribute of the Cassandra table can be a clustering key. This will have to be determined by the requirements engineer in consultation with domain experts.

There is a special case to (b). There can be certain categories for which the value of their attribute is taken from the system date and time. Such attributes get mapped to Cassandra's timestamp/timeuuid datatype. Assigning such a category/category attribute as a partitioning key creates as many partitions as the number of dates and time. To minimize the number of partitions we recommend to not use such attributes as the partitioning key. This is because such partitions provide no real value. Instead, we recommend that such a category attribute be assigned as a clustering key.

Applying our broad rules to the example taken in section 3 above, we get a table named waiting time. There are two 'computed from' information pieces, registration time and consultation time. Thus, these two become attributes of table waiting time. Further, category attributes: department code and department name belonging to category department also become attributes of the table.

Based on Rule 5, department and month are the partitioning keys. Let us say, department_code attribute represents the department. So, the partitioning key will be (department code, dateMonth). Registration time is the clustering key. The table is created with the CQL statement shown below in Table 1. Notice that there is a column dateMonth that represents month wise category.

Table 1: Table generated after applying Rules 1 to 5.

```
CREATE TABLE hospital.waiting_time (
        department_code text,
        dateMonth int,
        registration_time timeuuid,
        consultation_time timestamp,
        department_name text,
        PRIMARY KEY (( department_code, dateMonth),
registration_time)
)
```

The following table, Table 2, shows the sample of the data inserted.

Table 2: CQL commands for insertion of data.

```
INSERT INTO hospital.waiting_time
(department_code, dateMonth
,registration_time,consultation_time,department
_name) VALUES ('Sur',201901,33e90fc0-1efa-11e9-
8ca9-0fdba4c6d409,1547265902000,'Surgery');
INSERT INTO hospital.waiting_time
(department_code,dateMonth,registration_time,co
nsultation_time,department_name) VALUES
('Sur',201901,4317a0b0-1efa-11e9-8ca9-
0fdba4c6d409,1547267162000,'Surgery');
INSERT INTO hospital.waiting_time
(department_code,dateMonth,registration_time,co
nsultation_time,department_name) VALUES
('Med',201901,1599f7a0-1efa-11e9-8ca9-
0fdba4c6d409,1547264902000,'Medicine');
INSERT INTO hospital.waiting_time
(department_code,dateMonth,registration_time,co
nsultation_time,department_name) VALUES
('Med',201901,216427e0-1efa-11e9-8ca9-
0fdba4c6d409,1547265902000,'Medicine');
```

In order to clearly understand the partition based on department code and month of registration, let us look at the partition token obtained when rows are inserted into the table. Partition tokens can be obtained by CQL:

Select token (**department_code**, **dateMonth**) from hospital.waiting_time

The result of running the above statement can be seen in Figure 4. For all rows that have the same partition token Cassandra creates one row for each department and each month. So, for example, Medicine department and month of January will be one row. All the associated attributes will be stored as columns of the row.

```
system.token(department_code, datemonth)
------------------------------------------
                     3060662071838251383
                     3060662071838251383
                     5769990029507269337
                     5769990029507269337
```

Figure 4: Partition tokens obtained for the data inserted in Table 2.

Now, the aggregate function of the example of section 3 is to be mapped. Aggregate functions of the information model can be directly mapped to the aggregate functions of Cassandra. In our example, waiting time is calculated as the difference between registration and consultation time. Since there is no difference operator in Cassandra, we wrote a function, minus_time, to calculate the same. Once this was done, the aggregate function, avg, was applied and group by department code and month. Figure 5 shows the CQL code for the same.

```
cqlsh:hospital> select department_code, dateMonth
           ... ,avg(
           ... minus_time(
           ... dateOf(registration_time), consultation_time)
           ... )
           ... as Average
           ... from waiting_time
           ... group by department_code, dateMonth;

 department_code | datemonth | average
-----------------+-----------+-----------
             Med |    201901 | 974114348
             Sur |    201901 | 973038043
```

Figure 5: Obtaining the final aggregate information department wise and monthly.

Suppose we want information about waiting time but with a different categorization say, patient wise in addition to department wise and month wise. For the information model of figure 3, a completely new piece of information, compared to the earlier one, is generated. Thus, when we map this new information to Cassandra, a new table will be created with its own attributes and partition keys. In other words, each Cassandra table caters to one instantiation of the information model.

A tool to map the information model to Cassandra logical model based on the rules proposed is being developed.

5 CONCLUSION

There are traditionally two ways in which a DW system is implemented. One way is to directly use the data cube in a multi-dimensional database. Even though these systems give high performance, the databases are proprietary databases making a server niche. The other way to implement a DW is to use a relational database. Here, facts and dimensions are implemented as relational tables.

Relational databases suffer from the disadvantages of not being distributed, being sensitive to NULLs, not catering to all the different types of data that is to be stored, involving heavy join operations which impacts system performance.

ETL for a relational implementation is very time confusing and a slow process.

Therefore, we propose to use a NoSQL database. After all the information needs of the DW to-be have been identified by the requirements engineering phase, we propose mapping rules to take us to the logical model of NoSQL databases. For this, the information model is examined. Our preliminary work is for Cassandra, a column oriented database. Once the mapping is complete, OLAP operations can be performed.

Future work includes:

a) Defining the way in which OLAP operations will be implemented in Cassandra
b) Applying our mapping rules to a real-world example and evaluating our rules
c) Developing mapping rules for a document store. We have selected Mongodb as the database.
d) Developing mapping rules for XML databases.

REFERENCES

Boehnlein, M. and Ulbrich-vom Ende, A., 1999, November. Deriving initial data warehouse structures from the conceptual data models of the underlying operational information systems. In *Proceedings of the 2nd ACM international workshop on Data warehousing and OLAP* (pp. 15-21). ACM.

Böhnlein, M. and Ulbrich-vom Ende, A., 2000. Business process oriented development of data warehouse structures. In *Data Warehousing 2000* (pp. 3-21). Physica, Heidelberg.

Bonifati, A., Cattaneo, F., Ceri, S., Fuggetta, A. and Paraboschi, S., 2001. Designing data marts for data warehouses. *ACM transactions on software engineering and methodology*, *10*(4), pp.452-483.

Chevalier, M., El Malki, M., Kopliku, A., Teste, O. and Tournier, R., 2015, April. How can we implement a Multidimensional Data Warehouse using NoSQL?. In *International Conference on Enterprise Information Systems* (pp. 108-130). Springer, Cham.

Corr, L. and Stagnitto, J., 2011. *Agile data warehouse design: Collaborative dimensional modeling, from whiteboard to star schema*. DecisionOne Consulting.

Dehdouh, K., Bentayeb, F., Boussaid, O. and Kabachi, N., 2015, January. Using the column oriented NoSQL model for implementing big data warehouses. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)* (p. 469). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Duda, J., 2012. Business intelligence and NoSQL databases. *Information Systems in Management*, *1*(1), pp.25-37.

Giorgini, P., Rizzi, S. and Garzetti, M., A Goal-Oriented Approach to Requirement Analysis in Data Warehouses. *Decision Support Systems (DSS) jounal, Elsevier*, pp.4-21.

Inmon, W.H., 1995. What is a data warehouse?. *Prism Tech Topic*, *1*(1).

Inmon, W.H., Strauss, D. and Neushloss, G., 2010. *DW 2.0: The architecture for the next generation of data warehousing*. Elsevier.

Kimball, R., 1996. *The data warehouse toolkit: practical techniques for building dimensional data warehouses* (Vol. 1). New York: John Wiley & Sons.

Prakash, D. and Prakash, N., 2019. A multifactor approach for elicitation of Information requirements of data warehouses. *Requirements Engineering*, *24*(1), pp.103-117.

Prakash, N. and Gosain, A., 2003, June. Requirements Driven Data Warehouse Development. In *CAiSE Short Paper Proceedings* (Vol. 252).

Prakash, D., 2016. Eliciting Information Requirements for DW Systems. In *CAiSE (Doctoral Consortium)*.

Prakash, D., 2018, September. Direct Conversion of Early Information to Multi-dimensional Model. In *International Conference on Database and Expert Systems Applications* (pp. 119-126). Springer, Cham.

Prakash, N., Singh, Y. and Gosain, A., 2004, November. Informational scenarios for data warehouse requirements elicitation. In *International Conference on Conceptual Modeling* (pp. 205-216). Springer, Berlin, Heidelberg.

Santos, M.Y., Martinho, B. and Costa, C., 2017. Modelling and implementing big data warehouses for decision support. *Journal of Management Analytics*, *4*(2), pp.111-129.

Stonebraker, M., 2012. New opportunities for new sql. *Commun. ACM*, *55*(11), pp.10-11.