Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 53 (2013) 483 – 490

Procedia Engineering

www.elsevier.com/locate/procedia

Malaysian Technical Universities Conference on Engineering & Technology 2012, MUCET 2012
Part 4 - Information And Communication Technology

Storage Space Optimisation for Green Data Center

Nurul A. Emran[a], Noraswaliza Abdullah[a], Mohd Noor Mat Isa[b]

[a]Center for Advanced Computing Technology (C-ACT), Universiti Teknikal Malaysia Melaka,
Hang Tuah Jaya, 76100, Durian Tunggal, Melaka, Malaysia
[b]Malaysia Genome Institute,
Ministry of Science, Technology and Innovation, Selangor, Malaysia.

Abstract

The characteristic that determines a green data center is low amount of carbon footprint produced by its physical data storages. Increasing demand of data volumes in many data intensive applications call for additional physical data storages that are not only impractically large to maintain, but also contribute to the amount of carbon footprint produced. It is argued in this paper that, if storage space can be optimised to gain free spaces, the storage space requirement can be reduced. In this paper, a model to optimize database storage space by mining functional dependencies that are present among data sets is proposed. Sample data sets from the microbial domain have been used where data of large data volume raises storage space concern. The initial results of the implementation of the model that is necessary in designing a complete space optimization algorithm are presented.

Selection and peer-review under responsibility of the Research Management & Innovation Centre, Universiti Malaysia Perlis

Keywords: space-optimisation; functional dependency; green data centers.

1. Introduction

One prominent concern in the establishment of green data centers is to decrease carbon footprint and operating costs (e.g. cooling systems for data centers) by reducing the amount of physical data storages required. Scientific applications which rely on large of data volumes require physical data storages that are not only impractically large to maintain, but also contribute to inefficient power consumption.Within the context of scien- tific applications that require access to scientific databases, data volumes often be large enough for storage space requirements to become an issue that must be dealt by scientific data center providers. Expanding database storage is an option that data center providers could take in order to address the space issue, however this option leads to an increase in the amount of physical data storages (data servers) required. As more data servers are added, more electrical power is needed to run the additional data servers and to cooling-off those servers. The issue concerning data centers has been raised in a recent estimation which stated that the worlds data centers currently consume about 330 billion kWh of electricity every year, which is almost equal to the entire electricity demand of the UK [1]. In addition, power consumption that exceeds 100 billion kWh generate approximately 40, 568, 000 tons of CO2 emissions [2], [3], [4]. Thus, in establishing successful green data centers, adding more data servers is not an interesting option to choose in dealing with the storage space issue as this option leads to undesirable increase in power consumption and in CO2 emissions.

By optimising the available database storage required to store large data volumes, the requirements for physical data storages can be reduced. One way to reduce storage space requirement is by optimising the available database space. In fact, the need to optimise space is not new, as tools and techniques for this purpose provided by enterprise data storage vendors (such as Oracle [5], [6] and DB2 [7]) have been available in the market for about a decade. At the relational table level, data

1877-7058 © 2013 The Authors. Published by Elsevier Ltd.
Selection and peer-review under responsibility of the Research Management & Innovation Centre, Universiti Malaysia Perlis
doi:10.1016/j.proeng.2013.02.062

compression tools, for example, apply a repeated values removal technique to gain free space [6]. In addition, data deduplication techniques remove duplicate records in the table to gain storage space [8]. The idea behind these space optimisation solutions is to exploit the presence of overlaps (of values or records) within tables. Both of these techniques are performed at the level of whole tables. A key (though often unstated) assumption behind these optimisation techniques is that all columns can be exploited for space optimisation. Because of this assumption, knowledge of semantics of applications (i.e., how the columns are used) is ignored and as the consequence, data center providers need to bear unnecessary query processing overhead for frequent compression (and decompression) of heavily queried data.

The key lesson learnt from space optimisation techniques that are available in the market to date is that, space optimisation techniques that achieve space saving at both schema level and whole tables level are limited. In addition, space optimisation techniques that consider knowledge of semantics of applications have not been studied in depth. Because of these limitations, the two techniques described above unfortunately do not fully support solving the storage space issue faced by data center providers, where knowledge of how database is used must be considered for space optimisation. Therefore, an alternative space optimisation technique is needed to address the limitations of the existing techniques. This new, alternative technique is crucial to support data center providers in dealing with high storage space requirements that eventually call for unnecessary additional physical data storage.

Space optimisation will be useful in a data-intensive domain where storage space and establishing green data center is of concerned. Storage space requirement in the microbial genomics domain for example is driven by the need to con- duct various analyses (with a range of subjects coverage) and advancement in experimental tools provides a hint on the storage space issue faced by data-intensive application providers. Storing large data sets is not an issue if additional space can always be acquired by purchasing new disks every time the space needed exceeds the storage space's capacity. However, under a strict budget, purchasing new disks might not be feasible especially if frequent space expansion is required (i.e., due to the frequent addition of newly discovered data sets). Under these circumstances, if storage space can be optimized so that free space can be gained, the requirement for storage space can be reduced.

Space optimisation provides benefits not only when space is highly constrained, such as for handheld devices [9] but also when the concern is to optimise query response time. By reducing the space needed to store the data, we can reduce the time taken for input/output operations for the query, as data are stored in fewer blocks on the disk [10], [11]. In addition, space optimisation could ease the task of administering space expansion that usually requires new infrastructure, increased demand for utilities (for power and cooling), extra floor space, as well as additional staff [12].

In fact, the need to optimise the space is not new, as tools and techniques for this purpose, provided by enterprise data storage vendors (such as Oracle [13], [6] and DB2 [7]), have been available in the market for about a decade. At the relational table level, data compression tools, for exam- ple, apply a repeated values removal technique to gain free space [6]. In addition, data deduplication techniques remove duplicate records in the table to gain storage space [8]. The idea behind these space optimisation solutions is to exploit the presence of overlaps (of values or records) within tables. Both of these techniques are performed at the level of whole tables. A key (though often unstated) assumption behind these optimisation techniques is that all columns can be exploited for space optimisation. Because of this assumption, knowledge of semantics of applications (i.e., how the columns are used) is ignored. As a result, optimisation is usually performed at a whole table level.

The key lesson we learnt from these optimisation techniques is that, space optimisation techniques that achieve space saving at both schema level and whole tables level are limited. In ad- dition, space optimisation techniques that consider knowledge of semantics of applications are also limited. Because of these limitations, the two techniques described above unfortunately do not fully support solving the storage space constraint issue faced by data-intensive applications providers (such as the microbial genomics domain), where knowledge of how the data sets are used for the analyses must be considered for space optimisation. Therefore, in this paper, we set to propose a space optimisation technique that addresses the limitations which are barriers to deal with storage space issue which, the details of the technique will be presented in the next section.

2. The proxy-based approach

We propose a space optimisation technique called the proxy- based approach. Basically, the proxy-based approach method offers space saving through database schema modification, in particular by dropping attributes from the schema under consideration. The removal of the attributes, of course, will cause information loss and consequently will affect the queries that rely on those attributes. However, if the missing information can be retrieved from other attribute(s), the queries could still be computed using the smaller database. We use the term 'proxies' for attributes that substitute other attributes in the schema, which is inspired by proxies in other contexts with similar roles (e.g., in voting, a proxy is a person authorised to act on behalf of another [14]). We identified the proxies based on functional dependency relationship that can be observed among attributes in relational tables. An understanding of the space-accuracy trade-offs that the proxies could offer is required to facilitate the decisions in selecting which attributes can be deleted from the universe schema.

Therefore, answering the following questions regarding proxies are crucial before we can decide on its applicability:

- How do proxies contribute to space saving?
- How do we select the attributes to drop from the schema?
- What determines the amount of space saving that can be offered by proxies?

The idea behind the technique we propose is to achieve space saving through both database schema modification and exploitation of the presence of overlaps. Specifically, space saving through schema modification is achieved by dropping some attributes from the schema. If some attributes are dropped from the schema, the amount of space saved is roughly determined by the number of attributes being dropped and the number of tuples the table contains. For example, consider a table which consists of 100 tuples, with several attributes in its schema. If we drop an attribute from the schema, then the amount of space saved is 100 units of instances[*] (which is of course, is convertible to disk storage unit in bytes).

The question that arises is whether all attributes in the schema are droppable. To answer this question we need to understand the semantics of the application. As for the microbial genomics application, we need to understand how the data set is used in answering data set requests for the analyses. In particular, we need to know how attributes in the schema of the microbial database tables are used.

Consider the subject of interest in a microbial analysis gathered from their contributing sources is grouped into its population that is stored as a table. We define the generic schema of the population table as $P = \langle I, source, A \rangle$, where I is the set of identifier attributes, *source* is the attribute used to express the query predicates needed to extract individual record that is originated from specific microbial database and A is the set of attributes other than I and *source*.

Essentially, the identifier attribute is used in every query that retrieves the population individuals. The identifier attribute has an essential role in identifying population individuals. If we substitute the identifier attribute with a proxy, we need to assume that the same proxy is available in the source from which the person issuing the query extracted his/her data set under measure. However, in the situation where there is a lack of overlaps between the source from which the data set under measure is extracted and the integrated database, substituting the identifier attribute with a proxy will create a barrier to answer the queries. Therefore, because there is uncertainty as to whether the same proxy for the identifier attribute will be available in the source where the data sets is gathered, we regard identifier attributes as not droppable from the schema. This leaves the options of droppable attributes to those other than the identifier attribute.

Dropping attributes from their schema will cause informa- tion loss. Therefore, we must find other attributes that we refer to as proxies to substitute for the droppable attributes. The proxy and the droppable attribute must be related in some way to compensate for the information loss. A particular kind of relationship that can be observed among attributes in relational tables is a functional dependency (FD). In the literature, FD is described as a unique-value constraint, commonly used for relational schema design [15]. If an FD applies to a table, the value of an attribute on the table can be uniquely determined by the values of some other attributes [16]. An FD over a relation schema R is denoted by an expression $X \rightarrow A$ where $X \subseteq R$ and $A \in R$. X in the expression represents the set of attributes, known as 'determinant attributes' whose values can uniquely determine the values of A. The rule regarding FDs states that $X \rightarrow A$ is valid within relation r with schema R, provided that for all pairs of tuples t, $u \in r$ we have:

$$t[X[= u[X] => t[A] = u[A] \text{ where } A, X \subseteq R \quad [1]$$

If FDs within a table such as $X \rightarrow A$ hold, we suspect that if A is substituted by X , the information loss caused by dropping A from the schema can be compensated in full.

Therefore, based on our observations on the properties of FD, we say that, proxies for the droppable attributes can be found by discovering the relationships among attributes in the population tables where FD is present.

Discovering proxies based on FD is quite straightforward as they are usually key attributes within a relational table, especially if the information about the 'constraints' (i.e., PRIMARY KEY constraint) defined on the table is available. However, in the absence of the information about the types of constraints defined on the table; or in the case where FD-based proxies do not fall under the key attributes category, we need a way to discover them. An algorithm called TANE proposed by Huhtale et al. [16] is useful in our context to discover FDs. TANE was used in several applications such as semantic queries for query evaluation optimisation [17], accessing deep web results by relaxing the queries that fail to produce satisfactory results due to the constraints of web query interfaces [18] and, query rewriting that deals with nulls in query answers [19], [20].

The question that arises if proxies are accepted to substitute the droppable attributes is, how information loss as the result of dropping the droppable attributes can be compensated by the proxies? Space optimisation techniques described

[1] We regard each cell in a common relational table as an instance

earlier (e.g., based on compression) store the details of the removed overlaps in the meta-data in order to compensate information loss. These techniques implement algorithms that will retrieve the removed overlaps from the meta-data, every time a query is submitted against the compressed (or deduplicated) tables. To answer the question just raised, we adopted a similar way to compensate information loss where we use a proxy map to store the mappings between the values of the droppable attributes and the values of their proxies. For example, consider a and b are the values of a droppable attribute. A proxy map consists of the following mappings:

$$a \rightarrow \{1,2,3,4\},$$
$$b \rightarrow \{5,6,7,8\},$$

where the numeric values are the values of the proxy for the droppable attribute and the arrow illustrates a 'map to' relationship. Therefore, every time a query is submitted against a population table, the dropped attribute's values must be retrieved from the proxy map (for the query that has the dropped attribute in its predicate(s)).

2.1. Implementation Options of Proxy Maps

As storing the proxy map requires some space, the amount of space used by the proxy map must be taken into account, rather than considering space saving by the droppable attributes alone. Therefore, we define space saving as:

$$SpaceSaving = \frac{droppableAttrSize - proxyMapSize}{droppableAttrSize} \times 100, \quad (2)$$

where *droppableAttrSize* is the size of the droppable at- tribute, *proxyMapSize* is the size of the proxy map. The number of instances is used to represent the size of these variables, that is independent of any specific storage and file organisation system. One key criterion of a useful proxy, in terms of space saving, is that the amount of space required for the proxy map is smaller than the amount of space saved by dropping the attribute it substitutes for. Therefore, space saving through proxies will be of benefit if we could 'minimise' the space required to store the proxy map. The question that arises is how can we minimise the space required to store the proxy map? To answer this question, we need to identify the possible structures of a proxy map and compare them in terms of their size. Essentially, each proxy map must consist of the values of the droppable attributes and the values of the proxies that map to the values of the droppable attribute. We identified two possible structures for a proxy map as follows:

- A pure relational table: in its simplest form, a proxy map is in pure relational table structure for schema:<*droppableAttr, proxy*>. For this structure, each value of the droppable attribute will be mapped to exactly one value of the proxy. Because of this characteristic, each droppable attribute value that is substituted by multiple proxy values will be stored repeatedly in the table. For example, Table I shows an artificial proxy map in pure relational structure where, A is the droppable attribute and B is the proxy. Based on the table, 16 instances are stored within the proxy map.
- A multi-valued table: like the pure relational table, a proxy map in a multi-valued table structure has the schema: <*droppableAttr, proxy*>. However, unlike the pure relational table, each value of the droppable attribute will be mapped to a set of proxy values (that may be a sin- gleton set). Because of this characteristic, each droppable attribute value that is substituted by multiple proxy values will be stored only once in the table. This structure is also called as Vectorised dictionary based minimal DSM (VDMDSM) proposed by Rahman, Schallehn and Saake [21], which is a by-product of the decomposition storage model (DSM) by Copeland and Khoshafian [22]. The essence of this form of table structure in those works is to store all values of the same attribute of the relational conceptual schema relation together, for the advantage of performance (e.g., less database tuning required) and increased data independence and availability [22], [21]. For example, Table II shows an artificial proxy map in a multi-valued table structure, where A is the droppable attribute and B is the proxy. Based on the table, 10 instances are stored within the proxy map.

In comparison, the example shows that the size of proxy map in the multi-valued table structure (in terms of the number of instances stored) is smaller than the size of the proxy map stored in the pure relational structure. By adopting a multi- valued table structure in this example, we optimise the proxy table as a whole, by removing the repeated values (of the droppable attribute) which, if readers may notice, is a similar approach used by space optimisation techniques described earlier. Therefore, in the example, the space required to store the proxy map can be minimised by storing the proxy map in a multi-valued table structure. By minimising the storage requirement for the proxy map, we gain the benefit of space saving, which in the example above, is a saving of 6 instances. From the example, we say that storing the proxy map using a multi-valued table is useful in the situation where the droppable attribute values are mapped to multiple values of the proxy. To see whether a multi-valued table is useful by examining the size of proxy maps using real data sets, we conducted a small case

study in the microbial domain as presented in the next section.

Table 1. A proxy map in pure relational table

A	B
a	1
a	2
a	3
a	4
b	5
b	6
b	7
b	8

Table 2. A proxy map in a multi-valued table

A	B
a	1,2,3,4
b	5,6,7,8

3. Results of a case study of proxy maps implementation

We use microbial data sets taken from the Comprehensive Microbial Resources (CMR)[2] that were downloaded in the form of SQL dumps. In particular, we used three tables, namely evidence, taxon and bug attribute to represent our sample population tables. For brevity, we assume that the individuals of each population come from a single source (CMR). The schemas of these tables are as follows:

- Table 1: Evidence = <id, ev type, method,assign by>,
- Table 2: Taxon = <uid, genus, species, kingdom, ir1>,
- Table 3: Bug attribute = <id, att type, assignby>.

Evidence consists of the individuals of the microbes' evidence population, in which information about the type of evidence found about the presence of microbes, the method used to discovered them and the person (laboratory) who discovers them are stored; taxon consists of the individuals of the microbes' taxonomic population, in which information about microbes' genus, species, kingdom and intermediate ranking (of type 1) are recorded; bug attribute consists of the individuals of insects/bugs population where microbes were found, in which information about how the information is retrieved (attribute type) and the person (laboratory) who discovered them are stored. The underlined attributes are the keys for the tables. Each key attribute has been selected as the proxy (based on the presence of an FD) for all non-key attributes. The schema of the proxy tables are:

- from evidence:
  - <ev_type, id>, <method, id>, <assignby, id>.
- from taxon:
  - <genus, uid>, <species, uid>, <kingdom, uid>,<ir1, uid>.
- from bug_attribute:
  - <att type, id>, <assignby, id>.

We compared the size of proxy maps that are stored using a pure relational table structure and a multi-valued table structure. The sizes of the proxy maps were determined by counting the number of instances they contain. We use a 'working' proxy map to compute the size of proxy maps in both table structures, by issuing the following query against the population table, *p*:

$$\begin{aligned} \textit{workProxyMap} = \text{SELECT DISTINCT } & \textit{droppableAttr}, \\ & \textit{proxy} \\ \text{FROM } & p;, \end{aligned} \tag{3}$$

where *workProxyMap* is the working proxy map, *droppableAttr* is the droppable attribute and proxy is the proxy attribute.

The number of instances of the proxy map stored in the pure relational structure is counted based on the number of instances in the droppable attribute column and the proxy column retrieved by issuing the following queries:

$$\begin{aligned} droppableAttrInst &= \text{SELECT COUNT}(droppableAttr) \\ &\quad \text{FROM workProxyMap;} \\ proxyInst &= \text{SELECT COUNT}(proxy) \\ &\quad \text{FROM } workProxyMap;, \end{aligned} \tag{4}$$

where, *droppableAttrInst* is the number of instances in the droppable column of the proxy map, while *proxyInst* is the number of instances in the proxy column of the proxy map.

To count the number of instances of the proxy map stored in the multi-valued table schema, we issue the following queries against the working proxy map to get the number of instances in the droppable attribute column and in the proxy column:

$$\begin{aligned} droppableAttrInst &= \text{SELECT COUNT DISTINCT} \\ &\quad (droppableAttr) \\ &\quad \text{FROM } workProxyMap; \\ proxyInst &= \text{SELECT COUNT } (proxy) \\ &\quad \text{FROM } workProxyMap \end{aligned} \tag{5}$$

where, *droppableAttrInst* is the number of distinct (i.e., non- repeated) instances in the droppable column of the proxy map, and *proxyInst* is the number of instances in the proxy column of the proxy map.

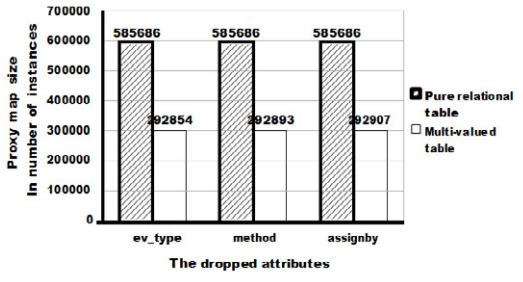
Thus, the size of the proxy map in both table structures is computed as:

$$proxyMapSize = droppableAttInst + proxyInst, \tag{6}$$

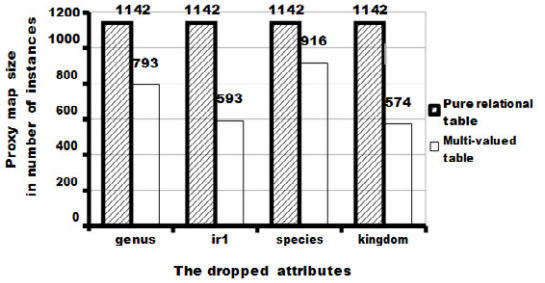
where *proxyMapSize* is the size of the proxy map.

Unlike *droppableAttrInst* for a pure relational proxy map (see Equation (3)), repeated droppable attribute val- ues are removed through the DISTINCT operator for *droppableAttrInst* in the proxy map stored in the multi- valued table (see Equation (4)). This is the distinguishing characteristic between the proxy map stored in a pure relational table and the proxy map stored in a multi-valued table.

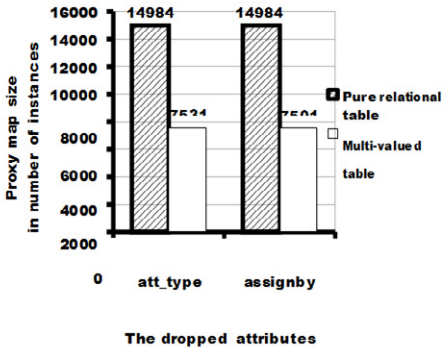
Figure 1(a), Figure 1(b) and Figure 1(c) illustrate the size of proxy maps by attributes in Evidence, Taxon and Bug_attribute respectively. These figures show a common result, which is the size of all proxy maps that are stored in a multi-valued table are smaller than the size of proxy maps in a pure relational structure. In addition, we can also see that for proxy maps in a pure relational table, the size of the proxy maps are the same for all attributes that belong to the same table. This shows that the size of proxy maps in pure relational tables is insensitive to the number of repeated droppable attribute values that map with multiple values of the proxy.



(a) Proxy map size by attributes from Evidence



(b) Proxy map size by attributes from Taxon



(c) Proxy map size by attributes from Bug_attribute

Fig. 1. A Comparison of Proxy Map Size by Table Structure.

In contrast, the size of proxy maps stored in multi- valued table is sensitive to the amount of repeated droppable attribute values that map with multiple values of the proxy. This means with a multi-valued table, the benefit of space saving is gained by removing the repeated droppable attribute values. Therefore, the result of this case study suggests the multi-valued table structure as a better structure to adopt than the pure relational table structure to store instances in the proxy maps.

4. Conclusion and future work

We presented an initial exploration of the proxy-based approach in which we answered the question of how the proxies could save storage space that will be of benefit for the establishment of green data centers. The results of a small case study using samples from the microbial genomics domain provide a comparison of the implementation options to store the proxy maps, which is crucial as an input in designing a complete algorithm for proxies selection in our future work. We would like to continue the work presented in this paper by answering other questions such as how proxies can be assessed, how can we correlate reduction in power consumption (and thus the amount of carbon footprint) and space saving; and how proxies can be implemented in systems where space optimisations and reducing carbon footprint are the goal.

Acknowledgment

The authors would like to thank the financial assistance provided by the Universiti Teknikal Malaysia, Melaka (UTeM) and the Ministry of Higher Education, Malaysia during the course of this research. The authors also acknowledge constructive comments received from Dr Suzanne Embury and the members of Information Management Group (IMG), School of Computer Science, The University of Manchester, UK; Dr Paolo Missier from The University of Newcastle, UK.

References

[1] G. Cook and J. V. Horn, “How dirty is your data? a look at the energy choices that power cloud,” Greenpeace International, 2011.

[2] S. Hazelhurst, “Scientific computing using virtual high-performance computing: A case study using the amazon elastic computing cloud,” in Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology. ACM, 2008, pp. 94–103.

[3] J. H. K. Kang, S. Cohen, W. Novak, and A. Peterson, “Feature-oriented domain analysis (FODA) feasibility study,” Technical Report, Domain Analysis Project, 1990.

[4] V. Kumar, “Algorithm for constraints-satisfaction problems: A survey,”AI Magazine, vol. 13, no. 1, pp. 32–44, 1992.

[5] Gredwid, “Green datacenters,” http://blogs.technet.com/b/nymciblog/archive/2008/03/21/datacenter- architecture-for-environmental-sustainability-green-datacenters.aspx,2008, [Online; accessed 25-Jun-2012].

[6] E. Lai, “Oracle pushes compression as cheaper database scale-up method,” Computerworld White Paper, 2008.

[7] C. Eaton, “Compression comparison to Oracle and Microsoft,” http://it.toolbox.com /blogs/db2luw/compression-comparison-to- oracle-and-microsoft-8871, 2006, [Online; accessed 11-May-2011].

[8] L. Freeman, “Looking beyond the hype: evaluating data deduplication so- lutions,” http://www.techrepublic.com/whitepapers/looking-beyond-the-hype-evaluating-data-deduplication-solutions/1294015, 2007, [Online; accessed 11-May-2011].

[9] T. Palpanas, N. Koudas, and A. Mendelzon, "Space constrained selection problems for data warehouses and pervasive computing," in Proceed- ings of the 15th International Conference on Scientific and Statistical Database Management (SSDBM). IEEE Computer Society, 2003, pp.55–64.

[10] R. Chirkova and C. Li, "Materializing views with minimal size to answer queries," in Principles of Database Systems (PODS). ACM, 2003, pp. 38–48.

[11] J. Naughton and S. Seshadri, "On estimating the size of projections," in International Conference on Database Theory, vol. 470. Springer, 1990, pp. 499–513.

[12] K. Ontrack, "The hidden costs of increasing data storage," http://www.ontrackdatarecovery.com/costs-increasing-data-storage/, 2006, [Online; accessed 11-May-2011].

[13] Oracle, "Advanced compression with Oracle database 11g release 2," http://www.oracle.com, 2009, [Online; accessed 11-May-2011].

[14] K. Petrik, "Participation and e-democracy how to utilize web 2.0 for policy decision-making," in Proceedings of the 10th Annual Interna- tional Conference on Digital Government Research: Social Networks: Making Connections between Citizens, Data and Government. Digital Government Society of North America, 2009, pp. 254–263.

[15] H. Molina, J. Ullman, and J.Widom, Database Systems: The Complete Book. Prentice Hall Press, 2008.

[16] Y. Huhtala, J. Ka¨rkka¨inen, P. Porkka, and H. Toivonen, "TANE: An efficient algorithm for discovering functional and approximate depen- dencies," The Computer Journal, vol. 42, no. 2, pp. 100–111, 1999.

[17] U. Nambiar and S. Kambhampati, "Mining approximate functional dependencies and concept similarities to answer imprecise queries," in Proceedings of the 7th International Workshop on the Web and Databases (WebDB). ACM, 2004, pp. 73–78.

[18] Y. Ma, D. Shen, Y. Kou, and W. Liu, "An effective query relaxation so- lution for the deep web." in 10th Asia-Pacific Web Conference (APWeb). Springer, 2008, pp. 649–659.

[19] G. Wolf, H. Khatri, B. Chokshi, J. Fan, Y. Chen, and S. Kambhampati, "Query processing over incomplete autonomous databases," in Proceed- ings of the 33rd International Conference on Very Large Databases (VLDB). VLDB Endowment, 2007, pp. 651–662.

[20] G. Wolf, A. Kalavagattu, H. Khatri, R. Balakrishnan, B. Chokshi, J. Fan, Y. Chen, and S. Kambhampati, "Query processing over incomplete autonomous databases: query rewriting using learned data dependencies," The VLDB Journal, vol. 18, no. 5, pp. 1167–1190, 2009.

[21] S. Rahman, E. Schallehn, and G. Saake, "ECOS: Evolutionary column- oriented storage," in 28th British National Conference On Databases, ser. Lecture Notes in Computer Science. Springer, 2011, pp. 18–32.

[22] G. P. Copeland and S. N. Khoshafian, "A decomposition storage model," SIGMOD Records, vol. 14, pp. 268–279, 1985.