Performance Evaluation of Natural and Surrogate Key Database Architectures[†]

Sebastian Link[1], Ivan Luković[2], Pavle Mogin[*)1]

[1]Victoria University of Wellington,
Wellington, P.O. Box 600, New Zealand
sebastian.link@vuw.ac.nz
pmogin@ecs.vuw.ac.nz
[2]University of Novi Sad, Faculty of Technical Sciences,
21000 Novi Sad, Trg Dositeja Obradovica 6, Serbia
ivan@uns.ac.rs

Abstract. Keys allow a database management system to uniquely identify tuples in a database. Consequently, the class of keys is of a great significance for almost all data processing tasks. There are two competing approaches to specifying keys in relational databases. These are natural and surrogate key architectures. A natural key is an attribute subset of the relation schema that is known to users. A surrogate key is a singleton attribute with system-generated values that are invisible to users. In the paper, we analyze the performance of these approaches with respect to different measures. These include the disk storage space, the number of indices, the update complexity, as well as the query data and expression complexity. In the analysis we use simple mathematical models and estimates of database performance. Our results provide new insights into the advantages and disadvantages of both approaches.

Keywords: natural key, surrogate key, database management system, database performance.

1 Introduction

Keys are a class of database constraints that have a great importance for maintaining a database in the consistent state. Currently, there are two competing approaches to defining keys. These are natural and surrogate key architectures.

A *natural key* is a candidate key that has a logical relationship to an attribute subset of the underlying relation schema. The existence of a natural key is known to users and businesses. It may be composed of a number of attributes, although there are numerous examples of single attribute natural keys. The main disadvantage of natural keys is the susceptibility to changes both in value and structure. The change in

[†] This research was supported by Victoria University of Wellington, Wellington, New Zealand Research Grant 85575

[*)] Corresponding author

the structure of a natural key usually results in serious problems in the database and application maintenance, since the change has to be applied in several places. As a consequence, the use of natural keys as primary keys is often disputed in industry.

A *surrogate key* is a singleton attribute whose values are (i) numeric, (ii) system generated and (iii) used for unique identification of tuples in a relation. Its existence and values are invisible to users. An important perceived advantage of using surrogates as primary keys is their immutability, which is a consequence of their separation from business logic. E. F. Codd in [3] defined a relational database architecture based on surrogates. Each relation schema has a surrogate attribute as its primary key. Surrogate primary keys are propagated to other relations as foreign keys.

In a recent survey of database implementation practices that we conducted in New Zealand and Serbia a prevailing number of respondents reported the use of a surrogate key database implementation architecture, where each relational database table has a surrogate primary key. The ease of implementation and particularly maintainability of database applications is often perceived as the main advantage of the surrogate key database implementation architecture.

We believe that natural keys should be specified whenever they represent business rules of the underlying application domain. Moreover, it appears to be difficult to argue against the claim that the ease of application development and the immutability against changes in the business rules are good reasons for the use of the surrogate key architecture. However, according to our best knowledge, no comprehensive analysis is available that studies the impact of natural and surrogate keys on the performance of the database. The main objective of this article is to address this research gap, and estimate the consequences of either implementation choice on:

- Update complexity,
- Query data complexity,
- Query expression complexity,
- Number of indices, and
- Disk storage occupancy.

To achieve the goal, we compare two border line approaches. In one of them, all relation scheme primary keys are natural keys or are composed of natural primary keys propagated from other relation schemes as foreign keys. In the other approach all relation schemes are associated with an invisible surrogate primary key, and only relation schemes produced by mapping entity types have also a natural candidate key.

Section two of the paper reviews related work. Section three briefly reports on the survey of database implementation practices in industry that we conducted in 2009. Section four examines and compares performances of two border line implementation architectures for two characteristic database structures. The final section presents conclusions and ideas for future work.

2 Related Work

Hall et al. [5] were among the first to consider the use of surrogates in database conceptual modelling. The motive to do so was the ambiguity that arises when a key value is modified. To illustrate the ambiguity, they consider the modification of a

personal number in a company's database and pose the question: "Have we recorded a change to someone's personal number or have we recorded his dismissal and replacement by a new employee (who happens to have the same name)?" To overcome the ambiguity, they introduce a collection of unique immutable objects which act as the representatives of the objects in the real world and call them *surrogates*. In the conceptual database, an entity is represented by its surrogate. A key is still needed to uniquely identify the surrogate, although a key value may be modified. The authors do not consider implications of surrogates on database performance.

E. F. Codd in [2] introduced new features into the relational data model to capture more meaning of data. One of these features is the surrogate primary key, where he builds on the results presented in [5]. Starting from the same position that users must be allowed to change values of keys that they have defined, Codd introduces a database wide domain, called *E-domain*, as the source of all surrogates. Each attribute defined on this domain has a recognizable name and permanent, database wide unique values. Further, he makes the use of user-controlled keys not compulsory. Since users are expected to define joins using surrogate keys, they have to be aware of their existence, but values have to be hidden from them. Codd does not consider the impact of the proposed extensions on database performance.

Wierenga et all in [9] gave a precise definition of object identification schemes. The authors introduced object identifiers which combine the ideas of natural and surrogate keys. We do not consider this concept in our article.

A search on Google returns almost one million hits for the key phrase *surrogate key*. Generally, these submissions briefly describe advantages and disadvantages of using natural or surrogate keys. In the following text, we present a submission that we have found most interesting.

Walker B. in [8] introduces a surrogate key architecture that relies on eight rules. These rules follow the lines defined in [3], with the exception that entity type tables contain beside surrogate also natural keys. Natural keys are needed to allow data searching. The main goal of this architecture is "to make the life easier for developers". The author doesn't present any quantitative analysis of the impact of surrogate keys on database performance.

3 A Survey of Database Design and Implementation Practices

During 2009 we conducted a survey of database design and implementation practices in industry. We made a questionnaire and asked a number of software companies and free lance database specialist from New Zealand and Serbia to answer it. A separate question asked participants to comment on the use of surrogate keys.

We received answers from five software companies and five independent database specialists. The majority of respondents reported to use a surrogate key architecture that closely follows the one defined in [2] and [5]. Only one software company answered that they exclusively use natural keys and claimed that they didn't experience any problems in doing so during over more than twenty years.

The ease of implementation and particularly maintainability of database applications were denoted as the main advantages of the surrogate key database

implementation architecture. The immutability was quoted as another important advantage of surrogate keys.

The common justification for using the surrogate key architecture was that application programmers do not want to think hard about database structure, particularly not to analyse composite keys, and write SQL queries involving joins on composite natural keys. These are even stronger reasons when developers have to fix a problem with a mission critical system that they have not seen before.

Tools like Oracle Designer and web application frameworks like Hibernate and Ruby on Rails stimulate the use of the surrogate key architecture by inserting surrogates as primary keys in the database description scripts either automatically or on designer's request.

Some of the respondents also referred to a relational database structure, very similar to one that would be obtained by mapping the ER diagram in Figure 1, as one that benefits considerably from using surrogate keys.

4 Performance Estimates

In the following text we perform a comparison of the natural key and surrogate key database implementation architecture using simple mathematical models and performance estimates. In our analysis, we consider the following performance criteria: the occupation of the memory space on disk, the number of keys and indices, the update complexity, and the query data and expression complexity. We investigate the performance differences of two ubiquitous database structures. These are the relational maps of a chain of higher-order relationship types and a binary first-order relationship type. Our ER notation follows the one introduced in [6].

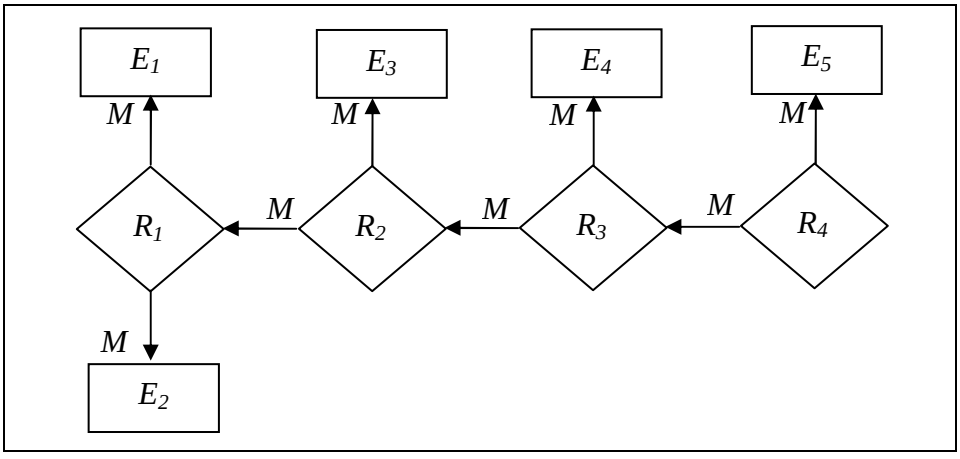

Figure. 1.

4.1 The Chain of Higher-Order Relationship Types

Let us consider the ER diagram in Figure 1. We denote an entity type as $E(W, K)$ where E is the name of the entity type, W is its set of attributes and K is a subset of W which is a key for E. We denote a relationship type by $R(C, V, K)$, where R is the name of the relationship type, C is the set of its components, V is its attribute set and K is its key. K is a subset of C and V, and the elements of K in C are called key components of R, while the elements of K in V are called key attributes of R. If the set C contains a relationship type, then R is a higher-order relationship type. Otherwise, R is called the first-order relationship type.

Let us denote a relation schema by $N(X, SCK, PK)$, where N is the relation schema name, X is the set of attributes, SCK is the set of candidate keys, and PK is the primary key. Let us also denote a referential integrity constraint by $N_1[Y] \subseteq N_2[PK]$, where N_1 and N_2 are relation schema names, $Y \subseteq X_1$, PK is the relation schema N_2 primary key, and Y and PK are union compatible sets of attributes.

Now we map the ER diagram in Figure 1 into two sets of relation schemes S and S' and two sets of referential integrity constraints I and I'. The mapping that results in S is a mapping based on natural keys, while the mapping that results in S' is based on the surrogate key architecture. If a relation schema from S or S' is obtained by mapping an entity type, we call it entity type relation schema. Otherwise, we call it relationship relation schema. Let nk_i ($i = 1, 2, \ldots, 4$) denote the natural primary key of the entity type E_i. An ER to relational mapping of the ER diagram above will produce the following set S of relation schemes:

S = {
$E_1(W_1, \{nk_1\}, nk_1)$,
$E_2(W_2, \{nk_2\}, nk_2)$,
$E_3(W_3, \{nk_3\}, nk_3)$,
$E_4(W_4, \{nk_4\}, nk_4)$,
$E_5(W_5, \{nk_5\}, nk_5)$,
$R_1(V_1 \cup \{nk_1, nk_2\}, \{\{nk_1, nk_2\}\}, \{nk_1, nk_2\})$, $R_2(V_2 \cup \{nk_1, nk_2, nk_3\}, \{\{nk_1, nk_2, nk_3\}\}, \{nk_1, nk_2, nk_3\})$,
$R_3(V_3 \cup \{nk_1, nk_2, nk_3, nk_4\}, \{\{nk_1, nk_2, nk_3, nk_4\}\}, \{nk_1, nk_2, nk_3, nk_4\})$,
$R_4(V_4 \cup \{nk_1, nk_2, nk_3, nk_4, nk_5\}, \{\{nk_1, nk_2, nk_3, nk_4, nk_5\}\}, \{nk_1, nk_2, nk_3, nk_4, nk_5\})$
},

and the following set I of referential integrity constraints:

I = {
$R_1[nk_1] \subseteq E_1[nk_1]$,
$R_1[nk_2] \subseteq E_2[nk_2]$,
$R_2[(nk_1, nk_2)] \subseteq R_1[(nk_1, nk_2)]$,
$R_2[nk_3] \subseteq E_3[nk_3]$,
$R_3[(nk_1, nk_2, nk_3)] \subseteq R_2[(nk_1, nk_2, nk_3)]$,
$R_3[nk_4] \subseteq E_4[nk_4]$,
$R_4[(nk_1, nk_2, nk_3, nk_4)] \subseteq R_3[(nk_1, nk_2, nk_3, nk_4)]$,

$R_4[nk_5] \subseteq E_5[nk_5]$,
}.

Let us transform the schema (S, I) into a schema (S', I') by:

1. Introducing surrogates sk_i as primary keys into the entity relation schemes E_i ($i = 1,\ldots, 5$), retaining their natural keys as candidate keys, and propagating surrogate primary keys into the corresponding relationship relation schemes,
2. Replacing relationship relation schema R_i ($i = 1,\ldots, 4$) composite natural keys by surrogates sk_j ($j = i + 5$) as primary keys, propagating sk_l ($l = 6, 7, 8$) into $R_{(l - 4)}$ as foreign keys, and forming candidate keys of foreign keys, and
3. Replacing composite natural keys by corresponding surrogate keys in referential integrity constraints from I.

An additional candidate key that is composed of propagated surrogates is needed in each relationship relation to guarantee uniqueness of tuples. By applying the above transformation, we obtain the following set of relation schemes:

S' = {
E_1' ($W_1 \cup \{sk_1\}$, $\{sk_1, nk_1\}$, sk_1),
E_2' ($W_2 \cup \{sk_2\}$, $\{sk_2, nk_2\}$, sk_2),
E_3' ($W_3 \cup \{sk_3\}$, $\{sk_3, nk_3\}$, sk_3),
E_4' ($W_4 \cup \{sk_4\}$, $\{sk_4, nk_4\}$, sk_4),
E_5' ($W_5 \cup \{sk_5\}$, $\{sk_5, nk_5\}$, sk_5),
R_1' ($V_1 \cup \{sk_1, sk_2, sk_6\}$, $\{\{sk_1, sk_2\}, \{sk_6\}\}$, $\{sk_6\}$),
R_2' ($V_2 \cup \{sk_3, sk_6, sk_7\}$, $\{\{sk_3, sk_6\}, \{sk_7\}\}$, $\{sk_7\}$),
R_3' ($V_3 \cup \{sk_4, sk_7, sk_8\}$, $\{\{sk_4, sk_7\}, \{sk_8\}\}$, $\{sk_8\}$),
R_4' ($V_4 \cup \{sk_5, sk_8, sk_9\}$, $\{\{sk_5, sk_8\}, \{sk_9\}\}$, $\{sk_9\}$)
},

and the following set of referential integrity constraints:

I' = {
$R_1'[sk_1] \subseteq E_1'[sk_1]$,
$R_1'[sk_2] \subseteq E_2'[sk_2]$,
$R_2'[sk_6] \subseteq R_1'[sk_6]$,
$R_2'[sk_3] \subseteq E_3'[sk_3]$,
$R_3'[sk_7] \subseteq R_2'[sk_7]$,
$R_3'[sk_4] \subseteq E_4'[sk_4]$,
$R_4'[sk_8] \subseteq R_3'[sk_8]$,
$R_4'[sk_5] \subseteq E_5'[sk_5]$
}.

4.1.1 Storage Occupation

In this section we estimate the storage occupation of relations in a natural key and a corresponding surrogate key database. To simplify the analysis we suppose that:

- Tuple fields are of a fixed size, and
- Relations are stored in files of fixed size blocks.

The relationship between a given number of tuples r of the size L and the number of blocks b of a size B is

$$b = \lceil r / f \rceil,$$

where f is the blocking factor, given as $f = \lfloor B / L \rfloor$.

We assume that entity relations tend to contain master record data, where typically $size(W) >> size(nk)$, while relationship relations tend to contain transaction data, where $size(V) = l*size(nk)$, and where $2 \le l \le 5$ may be a reasonable estimate, and l is a multiplier. Also, we expect that the number of tuples of each entity relation is by at least an order of magnitude smaller than the number of tuples of the associated relationship relation. So, we neglect the storage occupancy of entity relations in this analysis. Further, we assume the following relationship between numbers of tuples r_i in relations over R_j ($j = 1, 2, 3, 4$),

$$r_1 < r_2 < r_3 < r_4,$$

where $r_{j+1} > r_j$ ($j = 1, 2, 3$) by at least an order of magnitude. To simplify the analysis, we assume that all natural keys nk and all surrogate keys sk have the same size ($size(nk) = size(sk)$) of 8 bytes. Also, we suppose attribute sets V_j ($j = 1,\ldots, 4$) have the same size $size(V)$. Further, we assume the size of a block header $size(header)$ is 100 bytes, and the size of a disk block B is 8 KB. The block size of 8 KB represents its usual value today.

Table 1 displays number of disk blocks b occupied by instances over R_j and R_j' ($j = 1, 2, 3, 4$) having $r_1 = 10^4$, $r_2 = 10^5$, $r_3 = 10^6$, and $r_4 = 10^8$ tuples.

Table 1.

R_j	r_j	$size(V) = 2size(nk)$			$size(V) = 5size(nk)$		
		L	F	b	L	f	b
R_1	10^4	32	252	40	56	144	70
R_1'	10^4	40	202	50	64	126	80
R_2	10^5	40	202	496	64	126	794
R_2'	10^5	40	202	496	64	126	794
R_3	10^6	48	168	5953	72	112	8929
R_3'	10^6	40	202	4951	64	126	7937
R_4	10^8	56	144	694445	80	101	990100
R_4'	10^8	40	202	495050	64	126	793651

The table above shows that, under given assumptions, use of surrogate keys leads to a saving in number of disk blocks in database structures resembling chains of higher-order relationship types longer than one. In the case of a chain with three higher-order relationship types and $size(V) = 5size(nk)$, the total saving is approximately $200{,}000$ 8 KB blocks, or 20%. The amount of storage space saved increases with the length of the chain. But, if the chain contains only one higher-order relationship type, the use of surrogate keys leads to a small increase in the use of disk blocks.

4.1.2 Number Indices

The use of surrogate keys introduces an additional candidate key in all relation schemes. Contemporary database management systems automatically build a B-tree index for each primary key and unique constraint declared.

Since a natural keys and a surrogate key of the same relation may differ with regard to the number of components, the corresponding B-tree indexes may differ in the height and the number of nodes for the same number of tuples.

A B-tree is a dense index whose leaves are on the same path length from the root. A B-tree stores items of the form (*k*, *p*, *a*) in its nodes, where *k* is a tuple key or unique value, in a non leaf node *p* is a pointer to a node containing key values greater than *k*, otherwise *p* is null, and *a* is the address of the tuple with the key value *k*. Each B-tree node, except the root, may contain between *m* and *2m* (*m > 0*) items, where 2m is the maximum number of items in a node. The root may contain between *1* and *2m* items. All items of a node a sorted according to the ascending values of *k*. Each B-tree node has a header and, if it is a non leaf node, a pointer *p* to a node having smaller key values than the smallest key value in the node. Each B-tree node occupies a disk block of the size *B*. The value of *m* is calculated in the following way

$$m = \lfloor (B - size(p) - size(header)) \,/\, 2(size(k) + size(p) + size(a)) \rfloor.$$

The number of nodes *n* and the height *h* of a B-tree depend on the number of tuples *r*, the parameter *m*, and the extent to which nodes are filled. If all nodes contain *2m* items, the number of nodes is

$$n = r \,/\, 2m,$$

and the height is

$$h = log_{2m+1}(r + 1).$$

Bayer and McCreight (1972) reported that, in their experiments, B-tree nodes tended to be 67% full. Accordingly, the number of nodes of a B-tree may be estimated using the formula

$$n = \lceil r \,/\, l \rceil,$$

where $l = \lceil 2*0.67*m \rceil$, while the height of a B-tree may be estimated using the formula

$$h = \lceil \log_{l+1}(r + 1) \rceil.$$

Now, we compute the number of nodes and height of B-trees in our example natural key and surrogate key databases. We calculate the number of nodes and the height of natural key and surrogate key indexes for relationship relations only, since we do not expect indexes on entity relations to differ significantly, due to a smaller number of tuples and a large parameter *m*. To simplify the analysis, we assume that *size*(*nk*) (size of an entity type primary key), *size*(*p*), and *size*(*a*) are *8* bytes, *size*(*header*) is *100* bytes, and the size of a disk block *B = 8* kilo bytes. The following table displays parameters of the primary key B-tree and the other candidate key B-tree for instances over R_j and R_j' ($j = 1, 2, 3, 4$) having $r_1 = 10^4$, $r_2 = 10^5$, $r_3 = 10^6$, and $r_4 = 10^8$ tuples.

Table 2.

R_j	r_j	Primary Key B-tree				Other Candidate Key B-tree				Total nodes
		size(k)	*M*	*n*	*h*	*size(k)*	*M*	*n*	*h*	
R_1	10^4	*16*	*126*	*58*	*2*					*58*
R_1'	10^4	*8*	*168*	*44*	*2*	*16*	*126*	*58*	*2*	*102*
R_2	10^5	*24*	*101*	*718*	*3*					*718*
R_2'	10^5	*8*	*168*	*432*	*3*	*16*	*126*	*576*	*3*	*1008*
R_3	10^6	*32*	*84*	*8627*	*3*					*8627*
R_3'	10^6	*8*	*168*	*4314*	*3*	*16*	*126*	*5752*	*3*	*10066*
R_4	10^8	*40*	*72*	*1006442*	*5*					*1006442*
R_4'	10^8	*8*	*168*	*431332*	*4*	*16*	*126*	*575110*	*4*	*1006442*

The most important information conveyed by Table 2 is that, according to our suppositions, the heights of the surrogate primary key indexes and natural key indexes do not differ, although the surrogate keys have only one component, while the natural key may have several. This fact can be attributed to the logarithmic dependence of the tree height on the number of tuples and the large logarithm base (parameter *m*).

For the considered length of the chain of higher-order relationship types, the combined effect of the surrogate key index and the other candidate key index leads to a larger use of the memory space on disk.

4.1.3 Update Complexity

In defining semantics of updates and queries, we start from the position that users perceive the ER diagram in Figure 1 as the representation of their database and define their updates and queries to the database in terms of natural keys.

In this section, we consider effects of inserts, deletes, and modifications made to entity and relationship relations separately. The separation is made because entity relations contain master record data that are updated considerably less frequently than transaction data contained in relationship relations. While master record tables are expected to contain thousands of tuples, transaction tables are expected to contain millions of tuples. Accordingly, updates of relationship relations have a considerably greater impact on the overall database performance.

Inserts and deletes in entity relations with natural keys only are more efficiently performed than in entity relations with surrogate keys due to the need to update an additional index in relations with surrogate keys. The natural key modification is the operation where performances of the two database implementations differ most significantly. Since natural key values are visible to users, modifications of entity relation natural key values should be cascaded in natural key databases. A natural key modification propagates through relationship relations and may cause one or more index updates in all relations influenced by the update. A natural key modification in an entity relation with the surrogate key has the same effect as the modification of a non key attribute. It doesn't propagate, but it induces the update of an additional index. Namely, a DBMS performs a tuple modification by deleting the old version of the tuple and inserting the new one at the end of the last block in the physical memory

space allocated to the relation. Since the tuple address has been changed, all indexes have to be updated.

Now, we focus our analysis on inserts, deletes, and modifications of natural key relationship relations R_i and surrogate key relationship relations R_i' ($i = 1, 2, 3$). In the following discussion, we focus only on facts that mostly influence performance differences of update operations against the two database implementations. We discuss updates defined on natural key values first and then briefly comment on updates defined on non key attribute values.

Updates of instances over R_i are accomplished by issuing solely SQL INSERT, DELETE, and UPDATE commands. Updates of surrogate key relationship relations require the retrieval of appropriate surrogate key values before issuing SQL update commands. To make the last claim more obvious, we consider updates of instances of relations over R_3 and R_3'. Before an INSERT, DELETE, or UPDATE, an application program controlling updates has to execute the following two queries:

(1) SELECT sk_6 FROM E_1' NATURAL JOIN E_2' NATURAL JOIN R_1' NATURAL JOIN E_3' NATURAL JOIN R_2' WHERE $nk_1 = a_1$ AND $nk_2 = a_2$ AND $nk_3 = a_3$;

and

(2) SELECT sk_4 FROM E_4' WHERE $nk_4 = a_4$;

where a_i ($i = 1, \ldots, 4$) are natural key values.

Let us consider now an insert into a relation over R_3'. The new tuple has to contain an existing value of sk_7 to satisfy the referential integrity constraint $R_3'[sk_6] \subseteq R_2'[sk_6]$ and an existing value of sk_4 to satisfy the referential integrity constraint $R_3'[sk_4] \subseteq E_4'[sk_4]$. Since surrogate key values are hidden from users, the queries (1) and (2) have to be performed.

According to the above analysis, inserting into relations with composite natural keys has a definite advantage over inserting into relations with surrogate keys. An additional disadvantage for inserts into surrogate key relationship relations is the need to insert new keys into an additional B-tree.

Consider now the delete of a particular tuple from a relation over R_3'. This delete has to be defined either in terms of a sk_7 value, or a (sk_6, sk_4) value. Queries (1) and (2) retrieve the needed (sk_6, sk_4).

Deleting a tuple from relationship relations with composite natural keys has a definite advantage over deleting from relations with surrogate keys. An additional disadvantage for deletes from surrogate key relationship relations is the need to delete keys from an additional B-tree.

Consider now the modification of a particular tuple in a relation over R_3'. This modification has to be defined either in terms of a sk_7 value, or a (sk_6, sk_4) value. Queries (1) and (2) retrieve the needed (sk_6, sk_4). Again, due to the need to retrieve surrogate key values using queries and to update an additional B-tree, modifications take longer in relationship relations with surrogate keys.

Now, we discuss updates defined in terms of non key attribute values. If the search has been based on a non unique attribute value, the sequential scan algorithm is used to fined all tuples to be updated.. In this case searching in relations over R_i' is faster than searching in relations over R_i for $i > 2$, since search performance linearly

depends on the number of blocks. But each delete or modification also involves B-tree updates. Accordingly, there can not be expected a significant difference in update performance.

4.1.4 Query Data and Expression Complexity

In this section, we analyse the following queries:

a) (Query Q.a) Retrieve values of an attribute $A \in V_3$ for a conditional expression on the value k_1 of the natural key nk_1.
b) (Query Q.b) Retrieve values of an attribute $A \in V_3$ and an attribute $B \in W_1$ for the natural key value $nk_1 = k_1$.
c) (Query Q.c) Retrieve values of the natural key nk_1 for a conjunctive conditional expression on non unique attributes from the entity relation E_1 and relationship relations R_1, R_2, and R_3.
d) (Query Q.d) Retrieve values of an attribute $A \in V_3$ for a conditional expression on the value b of an attribute $C \in W_4$.

Above queries are selected to illustrate differences and similarities of the two architectures in answering queries, but do not pretend to give a definite answer which approach is generally better.

We use the following criteria to estimate the data complexity of a query:

- The number of joins, and
- The number of disk blocks of a relation.

We assume that the query data complexity stems mainly from the number of joins, and then from the number of disk blocks. The number of disk blocks determines the complexity of executing conditional expressions, and may influence the complexity of a join algorithm. If these two simple criteria are not decisive enough, then we also take into account complexity of the supposed join algorithm.

a) The query Q.a maps into

SELECT A FROM R_3 WHERE $nk_1 = k_1$; (Q.a natural)

in the natural key database, whereas the same query maps to

SELECT A FROM R_3' NATURAL JOIN R_2' NATURAL JOIN R_1' NATURAL JOIN E_1' WHERE $nk_1 = k_1$; (Q.a surrogate)

in the surrogate database. While the query Q.a is a single table query against the natural key database, it requires three joins when issued against the surrogate key database. The query is defined in terms of a natural key nk_1 value that is available in the natural key database from R_3 directly, and only from the entity relation E_1' in the surrogate key database. So, the query (Q.a surrogate) has to join the relations over E_1' and R_3' via intermediate relationship relations. Accordingly, the natural key database has a clear performance advantage over the surrogate database.

b) The query Q.b maps into

SELECT A, B FROM E_1 NATURAL JOIN R_3 WHERE $nk_1 = k_1$; (Q.b natural)

for the natural key database, whereas the same query maps to

SELECT *A*, *B* FROM R_3' NATURAL JOIN R_2' NATURAL JOIN R_1' NATURAL JOIN E_1' WHERE $nk_1 = k_1$; (Q.b surrogate)

for the surrogate database. The query asks for values of attributes from relations over E_1 and R_3 and maps into a multi table SQL expressions in both databases. The natural key database allows a direct join of relations over E_1 and R_3, while the surrogate key database requires joining all relations on the path from E_1' to R_3'. Accordingly, the natural key database has a clear performance advantage over the surrogate database.

c) The query Q.c maps into

SELECT nk_1 FROM R_3 NATURAL JOIN R_2 NATURAL JOIN R_1 NATURAL JOIN E_1 WHERE $B = b$ AND $A_1 = a_1$ AND $A_2 = a_2$ AND $A_3 = a_3$; (Q.c natural)

for the natural key database, whereas the same query maps to

SELECT nk_1 FROM R_3' NATURAL JOIN R_2' NATURAL JOIN R_1' NATURAL JOIN E_1' WHERE $B = b$ AND $A_1 = a_1$ AND $A_2 = a_2$ AND $A_3 = a_3$; (Q.c surrogate)

for the surrogate database. The query Q.c maps into SQL expressions with the same number of joins in both databases. Here, we estimate that a smaller number of blocks in the relation over R_3', compared to R_3 favours the performance of the query against the surrogate key database. The performance advantage stems from a more efficient linear search in the relation over R_3', but may be also influenced by a more efficient join algorithm.

d) The query Q.d maps into

SELECT *A* FROM R_3 NATURAL JOIN E_4 WHERE $C = c$; (Q.d natural)

for the natural key database, whereas the same query maps to

SELECT *A* FROM R_3' NATURAL JOIN E_4' WHERE $C = c$; (Q.d surrogate)

for the surrogate database. The query Q.d maps into a SQL expressions with a single join in both databases. Here, a slightly smaller number of blocks of E_4 compared to E_4' favours the efficiency of the linear search in the relation over E_4. On the other hand, the performance of the join is influenced by the smaller number of blocks of the relation over R_3' (in the case of a nested loop, sort-merge, or hash join algorithm), or a possibly smaller primary key B-tree height in the case of the single loop join algorithm. We estimate that the query Q.d has approximately the same performance in both databases.

Contrary to complaints reported in the survey, queries against the natural key database in examples above do not have a greater query expression complexity, since we used natural join. If we used inner joins, then the query (Q.c natural) would have a greater expression complexity than the query (Q.c surrogate).

4.2 First-order Relationship Type

Let us consider the ER diagram in Figure 2 and suppose the first-order relationship type R_1 is not involved in any higher-order relationship structure.

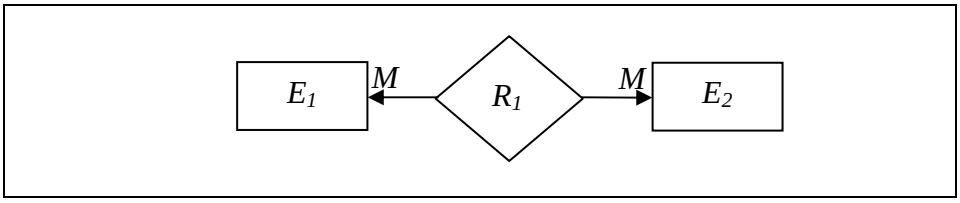

Figure 2.

The structure in Figure 2 maps to a natural key and surrogate key database schema having such sets of relation schemes *T* and *T'* that are subsets of relation schema sets *S* and *S'*, respectively. Instances over *T* and *T'* satisfying the corresponding referential integrity constraints have the following features:

a) The natural key relations occupy less storage space according to Table 1.
b) Relations over *T'* contain at least one additional candidate key and B-tree index.
c) B-tree indexes in relations over *T'* are of the same height as the indexes of the corresponding relations over *T*, but according to Table 2 occupy more storage space in total.
d) Inserts and deletes from an instance over R_1' take longer than the same operations against a corresponding instance over R_1, since:
   i. There is still a requirement to execute two SQL SELECT operations prior to performing actual update operations, and
   ii. The additional B-tree index needs also to be updated.
e) The modification of a natural key value takes longer, since the modification has to be cascaded to the relation over R_1.
f) Queries against the natural key database execute faster than queries against the surrogate database, since:
   i. Certain queries against the surrogate key database still require more joins, and
   ii. The relations of a natural key database occupy less storage space and that results in a more efficient linear search.

All claims above, except the last one, follow directly from the analysis performed in the section 4.1. To justify the claim f), let us consider the following query

(Query Q.c1) Retrieve values of the natural key nk_1 for a conjunctive conditional expression on non unique attributes from the entity relation E_1 and the relationship relation R_1.

The query Q.c1 is a restricted version of the query Q.c above, adjusted to fit a database structure that corresponds to Figure 2. As in the case of the original query Q.c, the SQL expressions of the query Q.c1 against both the natural key and surrogate key databases contain a join and two conditional expressions. Contrary to the case of the query Q.c, the natural key database structure occupies a smaller number of blocks, which leads to an estimate that the query against the natural key database may outperform the query against the surrogate key database.

The preceding analysis may be generalized to relational implementations of ER structures involving relationship types of any arity with associated entity types.

5 Conclusion and Future Work

In this paper, we have compared two relational database architectures: one that is based on natural keys with one that is based on surrogate keys. Utilizing simple mathematical reasoning we have obtained the following insights:

- The natural key architecture performs considerably better when updates to relationship relations are considered.
- The modification of natural key values is a potentially great disadvantage of the natural key architecture.
- If deep hierarchies of higher-order relationship relations are present, then the query performance of neither of the architectures is superior to the other.
- Queries against first-order relationship relations perform better with respect to the natural key architecture than they do with respect to the surrogate key architecture.

The majority of queries and updates in real life databases are defined in terms of natural keys. Accordingly, queries against relations in the surrogate key architecture require at least one join with an entity relation, but may involve several joins along the join path. Therefore, the natural key architecture can be expected to have a performance advantage due to the smaller number of joins needed.

If a query requires the same number of joins in both architectures, then the storage occupation has a deciding impact on the performance. In such cases, long chains of higher-order relationship relations favour the surrogate key architecture, while short chains of relationship relations favour the natural key architecture.

The natural key architecture does not benefit considerably from indexes on foreign keys. If databases are update-intense, then it may be advisable not to use them. In such situations, e.g. in OLTP databases, the performance advantage depends on the ratio of the number of modifications of natural key values and the number of updates of relationship relations.

References

1. Bayer R. and McCreight E. (1972): Organization and Maintenance of Large Ordered Indexes, *Acta Informatica, Vol.1, 1972, pp.173-189.*
2. Codd E. F. (1979): Extending the Relational Data Model to Capture More Meaning, *ACM TODS*, 4(4): 397-434.
3. Date C., Darwen H., and Mcgoveran D.(1998): Relational Database Writings 1994-1997, *Addison Wesley, 1998*.
4. Fagin R. (1981): A Normal Form for Relational Databases That Is Based on Domians and Keys, ACM TODS 6(3): 387-415
5. Hall P., Owlett, J. and Todd, S. (1976). Relations and Entities, *Proc. IFIP TC-2 Working Conference on Modelling in Data Base Management Systems*, G. M. Nijssen (ed.), pp 201-220, Horth-Holland.
6. Thalheim B. (2000). Entiry-Relationship Modeling – Foundation of Database Technology, Springer Verlag, Berlin, New York , 2000.

7. Toman D., Weddell G. E. (2008): On Keys and Functional Dependencies as First-Class Citizens in Description Logics. J. Autom. Reasoning 40(2-3): 117-132.
8. Walker B. (2006) *Why Use Surrogate keys*, SearchSQLServer.com, http://searchsqlserver.techtarget.com/.../0,295582,sid87_gci1156604,00.html Accessed 05.01.2010.
9. Wieringa R., De Jonge W. (1995): Object Identifiers, Keys, and Surrogates: Object Identifiers Revisited. Theory and Practice of Object Systems, 1(2):101-114.
10. Wikipedia, Surrogate Key (http://en.wikipedia.org/wiki/Surrogate_key). Accessed 05.01.2010.