# Logical and Physical Data Structures for Very Small Databases

Cristiana Bolchini, Fabio Salice, Fabio A. Schreiber, Letizia Tanca

Politecnico di Milano, Dip. Elettronica e Informazione

Proc. SEBD'02
Portoferraio, June 2002

pp. 337-344

# Logical and Physical Data Structures for Very Small Databases

Cristiana Bolchini, Fabio Salice, Fabio A. Schreiber, Letizia Tanca

Politecnico di Milano, Dip. Elettronica e Informazione
P.zza L. da Vinci, 32, I20133 Milano, Italy
{bolchini, salice, schreibe, tanca@elet.polimi.it}

**Abstract.** The widespread adoption of portable devices and of embedded processors for professional embedded systems, and the technology underlying such devices raise new challenges. We propose physical and logical data structures to provide efficient, energy saving and endurance preserving access to data, managed by means of a DBMS. The paper relates on a project studying innovative data structures at the logical and physical level, to be adopted in very small databases hosted by portable devices/embedded systems, characterized by Flash EEPROM storage. The task is in fact part of a complete methodology for designing very small databases.

## 1 Introduction and Motivation

Today portable devices (PDA, palm, smartcards, cell phones), from PC downwards, are becoming widespread, and traditional applications are scaled down in order to cope with the limited resources of such devices in terms of processing power and performance results. Many of these applications rely on information systems, whose data may be distributed among different traditional data sources, but can be partially stored on the portable device, as a part of a larger database. As a result, the problem of designing databases for this class of devices is twofold. On the one hand, we should start to conceive DBMSs as small as possible, where the classical ACID properties should be at least partially guaranteed, taking into account the new features of the medium; on the other hand, the distributed application design must indicate which parts are located on traditional physical mediums (such as large scale computer systems) and which are located onto miniature devices. For latter data, appropriate logical and physical data structures should be defined, allowing for an efficient (integrated) access to data, both from the performance and the power consumption viewpoints, the latter being one of the specific critical aspects of this class of devices.

Our project [1,2] proposes a design methodology supporting the database designer in the identification and the localization of such information areas at conceptual schema definition time; it also proposes a logical design phase which is, more than traditionally, integrated with the physical design phase, and where tables are analyzed w.r.t. the envisaged access types, information volatility, user permissions and protection mechanisms; output of this phase is the suggestion of the most suitable data structures to be adopted to physically implement the database relations. The rest of the paper is organized as

follows. Section 2 introduces the complete design methodology, briefly describing its steps. Section 3 focuses on the proposed implementation for VSDB relations, discussing the logical and physical data structures allowing a satisfactory performance/cost trade-off in accessing stored data. In section 4 an evaluation of the proposed data structures is made w.r.t. performance and power consumption.

## 2 The proposed methodology

Fig. 1 briefly describes a complete framework for designing very small databases for embedded processors and portable devices.

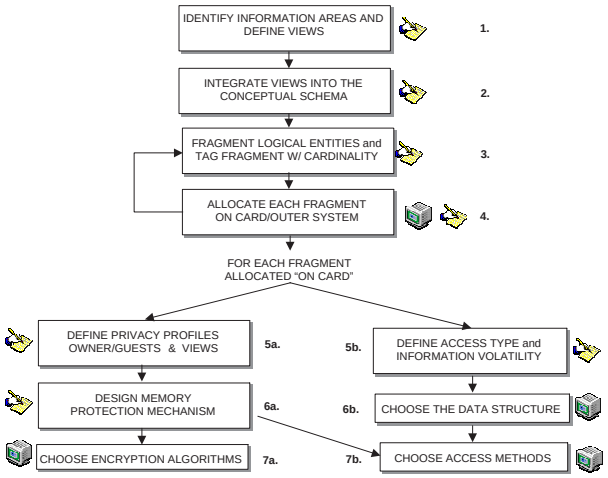


**Fig. 1.** The proposed methodology for supporting the database designer: steps denoted by a hand writing icon are a burden of the designer and an input to the procedure, while those denoted by PC icons can be automated.

The methodology includes a common track, derived from distributed/federated database design methodologies [3,4], which takes care of the conceptual and logical aspects; the lower part deals with "on chip" features: the left branch concerns privacy/security aspects while the right branch concerns the physical memory data structures and algorithms.

In the sequel we shall mainly focus on the problems related to the right branch; for a detailed description of the privacy related matters, refer to [5].

In steps 1 and 2, views are defined for the relevant information, at a conceptual level, singling out homogeneous information areas. Views are integrated into a global conceptual schema solving possible representation and semantic conflicts. In steps 3 and 4, fragmentation of logical entities is made in order to separate "first need" information - to be stored on the smart card - from the other fragments - to be stored in

other sites of the Information System. A close estimate of the fragment cardinalities is to be made at this step. Once provided an entity allocation hypothesis (on the card vs. on the main DBMS) a check is to be made on the smart card storage capacity: possibly fragmentation and allocation criteria shall be reconsidered. The other steps of the right branch, we have been focusing our attention on, are the following. 5b: expected dominant type of performed operations (select, update, ) and data volatility are estimated for each relation in the database; 6b: for each relation the most suitable physical data structure is selected based on the information annotated at step 5b; 7b: access methods are chosen for the data structures selected at step 6b depending on the constraints introduced at step 6a. Steps 6b and 7b are totally automatic, and a prototype tool is under development.

The following section details the introduced logical and physical data structures.

## 3 Data models, logical and physical data structures

The right-side branch of our methodology aims at identifying the most convenient logical and physical data structures for managing the information to be stored, retrieved and manipulated through the database. The efficiency of the solution is evaluated in terms of: a) access time to perform the desired operation, b) amount of required storage, and c) power consumption required by the desired operation. The technology behind the class of devices we are considering has a relevant impact on the above mentioned aspects, and will be briefly discussed in the sequel.

### 3.1 Technology issues

When considering portable devices, the permanent memory medium (the storage) is a nonvolatile memory support, EEPROM or Flash EEPROM. The technology of this class of memory is such that, for Flash-EEPROM (from now on called simply *Flash*), `write` data operations are `program`ming operations that can only be performed if the target location has either never been written before, or has been previously erased. A further complication and space/performance cost arises from the fact the `erase` operation works only at block level granularity, not at record level granularity. *Flash* memories are divided into blocks and each block can be erased separately. `read` and `program` operations work at the single bit/byte granularity; any byte (word) can be read or programmed separately. Endurance is a critical factor as well; each erasure has an impact on the life of the device. If the number of expected `erase`/`program` cycles (100,000 may cover a 5 years period) can seem satisfactory for a portable device, the same does not hold true for embedded systems for specific application where such a period is far from the lifetime of the entire system. Do note that, once the device quality is deteriorated, the reliability of the device is jeopardised.

Thus, in a Flash memory, in order to achieve good performance and a long endurance it is necessary to reduce the number of data modifications. Since *update*, *delete* and *insert* operations are required and inevitable, the sequel of the paper focuses on the minimization of the cost associated with the required operations with respect to the mode data have to be managed and stored.

| Characterisation | Values |
|---|---|
| Read (nsec) | 20 to 150 |
| Program (msec/byte) | 2 to 7 |
| Erase (s/block) | 0.7/0.8 |
| Cell Size (mm2 - 0.6 mm tech) | 6,4 |
| Endurance (program, erase) | 100.000 |

**Table 1.** Flash-EEPROM technical characteristics

### 3.2 Data structures for very small databases

The proposed approach [6] targets very small databases characterized by small volumes of data to be managed, an assumption deriving from the class of applications suitable for portable devices. Based on this hypothesis, we tried to identify a limited number of what we call “logical/physical data structures”, i.e. intermediate data structures that should be chosen to implement each database relation (without loss of generality, we shall refer to the relational model of data [1]). In the description of such data structures we refer to our running example, the Portable Internet DataBase (PIDB), storing personal information for internet access (see Fig. 2).

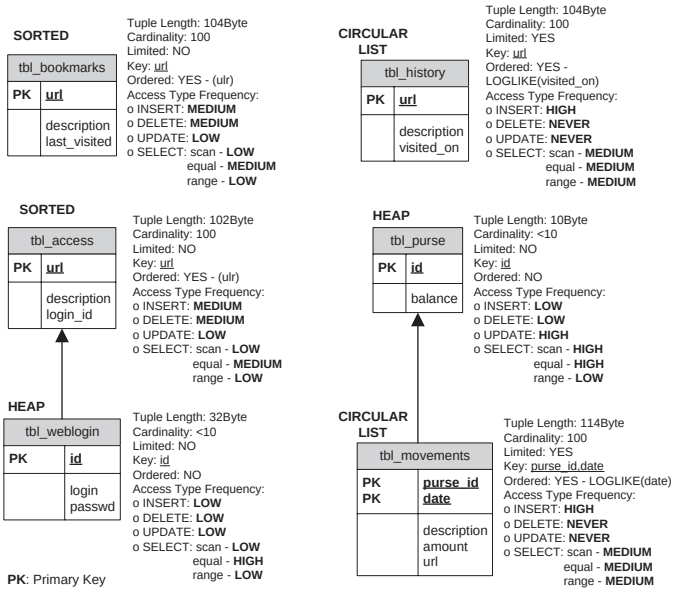


**Fig. 2.** The Portable Internet DataBase: Table Annotation and adopted data structure

A **Heap** relation is used to store a small number of records (generally less than 10), unsorted, typically accessed by scanning all records when looking for a specific one;

example of this kind of data is the login/password data in the PIDB database, storing different registered user identities.

**Sorted** relations, characterized by a medium (≅100 - ≅1000 records) cardinality are used to store information typically accessed by the order key. The idea here is to impose an upper bound to the number of records that can be inserted based on the complete size of the (fragment of) table. Once the upper bound is reached, the user will have to delete (or store externally) a record before adding a new one. With respect to the running example, URL Bookmarks can be managed by means of sorted data.

**Circular list** relations, still characterized by the same cardinality of the previous data type, are again suitable to manage a fixed number of log data, sorted by date/time; in this case, once the maximum number of records is reached, the next new record will substitute the oldest one. In the PIDB example, data logs of the last m URLs most recently visited and the last *n* payments in the E-purse can be stored by means of circular lists.

**Multi-index** relations, not belonging to the previously defined categories, are used to manage generic data, typically when the need is to access efficiently relations by multiple keys. This is the only data structure we propose which resembles the classical data structures used in DBMS's.

Our methodology (step 5b) requires the designer to tag each entity of the logical ER schema for the database under consideration with the following information:

- Tuple length (in bytes) and Expected Relation Cardinality (eventually specifying an upper bound to the number of records to be allowed (e.g., 100 entries for the visited web sites history).
- Presence of a sorting field, specifying if the field is a time field leading to a log-like
- Expected composition of the set of operations on data: *insert*/*delete*/*update*/*select* (this last one further classified in a full select - *scan* -, select with equality - *equal* - and select with range - *range* -).

The expected composition refers to the frequency of operations estimated with respect to one another: it cannot be a precise value, rather a relation between the possible operations. For instance, consider the `tblhistory` relation. The user can say that the dominant operation will be the *insert*, usually no *delete* and no *update*. The other common operation is the *select*, assuming an equal distribution in the three identified selection schemas.

### 3.3 Physical implementation: memory management

The technology behind Flash memories and their constraint on data erasure introduces a significant impact on the *delete* and *update* operations, also affecting *insert* operations in sorted relations. In fact, when the stored data need to be modified, at least one memory block needs to be re-written, implicitly requiring a copy of the content in the RAM, an erasure of the Flash block and a write-back (from RAM to Flash) of the modified content of the block (*dump*/*erase*/*restore* DER sequence). Eventually, this operation may involve also the adjacent memory blocks.

In order to reduce the number of modifications requiring the erasure of the Flash memory, we introduced an additional information associated with each record of a relation:

- **valid bit** to indicate if the record has been programmed;
- **deleted bit** to indicate if the record has been logically (but not physically) deleted.

The use of the *valid bit* is used to physically discriminate empty records from written ones, and is required for the particular memory technology when memory is programmed in a non-sequential fashion. Though this bit is essential when memory is managed in a non-sequential fashion, the use of a validity bit is more general and involves the dilemma of distributed and concentrated control on the stored data. In particular, the valid bit is a "distributed" control since each valid record is directly distinguishable from the others while the use of an end-address (register) is a "concentrated" control since it univocally identifies the end of the record list. The concentrated control is a space-aware but energy-time consuming approach since the end-address value needs to be updated every time the list is modified, while the *dump*/*erase*/*restore* sequence. The *deleted bit* is used to allow the system to reduce the number of Flash memory erasures by programming the bit of the corresponding record and to postpone the physical expunging to a later moment. Do note that the DER sequence deeply affects performance (due to the time required for the data "dump"), power consumption and the storage endurance. The introduction of the *deleted bit*, coupled with a not-sequential management of the physical memory leads to reduce the necessity to erase blocks, at the cost of an increase in the amount of required memory and a more complex management policy, as discussed in the following.

When dealing with data sorted with respect to a field, insert and delete operations have a significant overhead due to the necessity to maintain the data ordered; furthermore the operation may affect a single block or multiple blocks if the relation data is distributed over several blocks. The proposed data structure aims at a) confining block involvement in data manipulation and b) minimising block erasure. This goal is achieved by means of introducing a number of dummy records per block (Fig. 3a); such records may be either localised at the end of the block or they may be distributed through it.

In the latter case records are stored not adjacently: a hashing function is used to allocate records in the block so that future insertion do not always cause a re-organisation of previously introduced records (Fig. 3b). The hashing function may be implemented either in software or in hardware; in this case the *valid bit* is mandatory to determine which records are actually programmed and which are not. The use of the concentrated dummy records aims at preventing multiple blocks involvement when records need to be shifted up or down following a delete or insert operation (intra-block erasures). The distributed dummy records solution further limits inter-block erasures.

The *deleted bit* has the same function as above (Fig. 3c).

The combined use of dummy records and *deleted bit* is useful in sorted relations, the use of the *deleted bit* technique alone is suitable for circular lists and possibly heap relations, at the cost of additional space requirement w.r.t. the minimum possible amount of memory. The database annotation performed at step 5b of the entire methodology

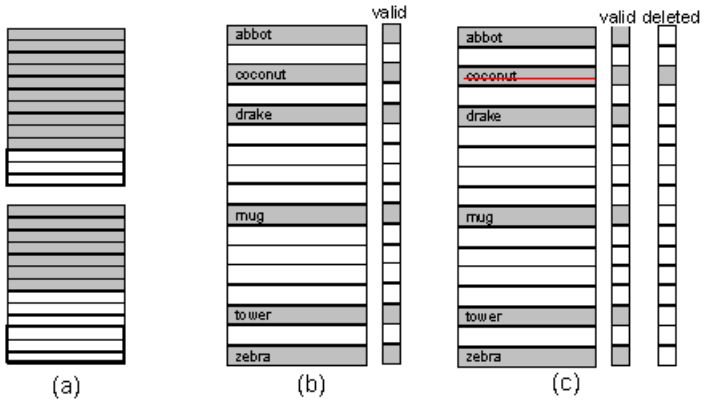


**Fig. 3.** Use of dummy records a) concentrated at the end of the block - bold frame - or b) distributed through the block. c) Use of distributed dummy records and *deleted bit*

allows the association of the most convenient data structure, as above discussed. Current research is investigating the impact and costs of such data structures on generic relations not falling in previous cases.

## 4 Structures evaluation and concluding remarks

The proposed data structures together with the possible physical implementation, have been analysed and compared with a "naive" implementation, both w.r.t. performance and power consumption. Costs have been estimated in terms of the total amount of necessary memory due to the additional bits associated with each record, and the dummy records; algorithm complexity and the consequent execution requirements (time and power consumption) have been also estimated. The first results show that the combination of the two strategies dummy records (requiring the use of the *valid bit*) and *deleted bit* allows to reduce the average time to perform the operations on data and to significantly limit the number of erasures. It is worth noting that, with the proposed structures, *select* operations have an overhead due to the presence of the empty records that slow the scanning of the records. A simulator has been developed to evaluate all parameters and to compare different solutions; currently our effort is devoted to the analysis of generic relations and to the enhancement of the application of these data structures in different moments of the database life, depending on the volatility of data. Table 2 reports a summarization of the experimental results carried out for evaluating the proposed data structures, for each relation. A synthetic workload made of operations evenly distributed between *insert*, *delete* and *update* was used, increasing memory occupation to analyse the impact on block erasures and the amount of information transmitted on the bus, two significant aspects affecting performance and power consumption.

The setup for the experiments is: 4Kbyte Flash Memory blocks, a 32 KByte RAM and a 128 byte record size for the relations; then number of Flash blocks and the number

| Data structure | Strategy | Block Erasures | | | Transmitted Bits on Bus | | |
|---|---|---|---|---|---|---|---|
| | | 10%-30% | 40%-60% | 70%-90% | 10%-30% | 40%-60% | 70%-90% |
| Heap | Simple | 1 | 1 | 1 | 1 | 1 | 1 |
| | Delete bit | 0 | 0.38 | 0.98 | 0.38 | 0.54 | 1.00 |
| Sorted | Simple | 1 | 1 | 1 | 1 | 1 | 1 |
| | Delete bit | 0.83 | 0.68 | 0.79 | 0.74 | 0.71 | 0.77 |
| | Dummy conc. | 0.83 | 0.51 | 0.44 | 0.74 | 0.57 | 0.45 |
| | Dummy dist. | 0.10 | 0.12 | 0.24 | 0.03 | 0.06 | 0.22 |
| Circular List | Simple | 1 | 1 | 1 | 1 | 1 | 1 |
| | Delete bit | 0 | 0 | 0.05 | 0.07 | 0.07 | 0.15 |

**Table 2.** Experimental results: occurred block erasures and transmitted bytes on the system bus w.r.t. "the naive", no *deleted bit*, no dummy records solution.

of records per relation differ in the various situations, depending on the data structure being simulated.

The first part of table 2 reports the ratio between the number of block erasures occurring when adopting the specified implementation and the number of block erasures when using a "naive" implementation. As an example, consider the *sorted* relation with a half full memory (50%): with the concentrated dummy solution the workload causes only half the number of block erasures w.r.t. the "naive" implementation. Similarly, in the second part of the table we reported the ratio between the number of bits trasmitted on the data bus for each proposed approach w.r.t. the "naive" implementation.

As it can be noted, the achieved results show an improvement in the Flash memory access both in terms of erasure operations and read/written bits, elements affecting the system performance and power consumption.

Our plans for the future include the development of a full-fledged methodology, designed to be computer supported in as many steps as possible. We will also further investigate strategies for power saving and performance enhancement.

## References


1. P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone. Database systems. McGraw-Hill (2000)
2. C. Bobineau, L. Bouganim, P. Pucheral, P. Valduriez. "PicoDBMS: Scaling down Database Techniques for Smartcard". Proc. 26th Int. Conf. on Very Large Databases (VLDB), (2000) pp.11-20.
3. S. Ceri, G. Pelagatti. Distributed Databases: Principles and Systems. McGraw-Hill (1984)
4. Smartcard adoption for ID application in the Italian Government, `http://www.cartaidentita.it/cie/reader/index.html`, 2002.
5. C. Bolchini, F. A. Schreiber. "Smart card embedded information systems: a methodology for privacy oriented architectural design". Data & Knowledge Engineering, Elsevier Science, Amsterdam, vol. 41 (2-3) (2002) 159-182.
6. C. Bolchini, F. Salice, F. Schreiber, L. Tanca. "Logical and Physical Design issues for Smart Card Databases". Technical Report no. 2001.68, Politecnico di Milano (2001).