International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:11, 2009

UB-Tree Indexing for Semantic Query Optimization of Range Queries

S. Housseno, A. Simonet and M. Simonet

***Abstract*—**Semantic query optimization consists in restricting the search space in order to reduce the set of objects of interest for a query. This paper presents an indexing method based on UB-trees and a static analysis of the constraints associated to the views of the database and to any constraint expressed on attributes. The result of the static analysis is a partitioning of the object space into disjoint blocks. Through Space Filling Curve (SFC) techniques, each fragment (block) of the partition is assigned a unique identifier, enabling the efficient indexing of fragments by UB-trees. The search space corresponding to a range query is restricted to a subset of the blocks of the partition. This approach has been developed in the context of a KB-DBMS but it can be applied to any relational system.

***Keywords*—**Index, Range query, UB-tree, Space Filling Curve, Query optimization, Views, Database, Integrity Constraint, Classification.

I. Introduction

PERFORMANCE enhancement is an important research area in the database domain especially when the DBMS deals with huge volumes of data. This problem has become crucial with the advent of applications/systems like Data Warehouses, Geographical Information Systems (GIS), spatial databases, multimedia databases, etc. Several techniques have been proposed and used to improve the performance of DBMS at the software level. Among these methods, there are data clustering, indexing data structures, query optimization, buffering, etc. Since a physical organization of data based on efficient indexing data structures with adapted query processing is one of the keys to efficient data retrieval, an important number of works in this domain have been proposed and implemented. There are two approaches to organize physically data on a secondary storage:

1) indexing based on a single attribute, e.g., hashing techniques, binary-tree, B-tree family [3], and
2) indexing based on multiple attributes, known as *multidimensional indexing,* e.g., multi-dimensional extensible hashing [7] R-trees [21], X-tree [8], Grid files [19], EXCELL [18].

This paper deals with the second approach.

The indexing method presented in this paper was designed for an object data model that aims at unifying databases and knowledge bases [16]. This model has been implemented in the KB-DBMS prototype Osiris. As a DBMS, it is based on views defined by logical constraints on attributes; as a KBMS it performs instance classification on every object in the database.

However, the indexing method presented in this paper can be applied to any relational system provided it is possible to build a partitioning of the data (object) space into disjoint clusters.

In the Osiris KB-DBMS, a static analysis of the object data model enables the system to build a partitioning of the object space into disjoint blocks. Each block covers a portion of the object space. Instead of indexing directly the objects, the system indexes the blocks. For each query, the smallest set of indexing blocks that « contains » the query can be determined.

The problem addressed in this paper is that of representing the object space through the disjoint blocks, in order to support efficient access to the objects of a query. Blocks are by nature multi-dimensional. A block is a hyper-rectangle in a N-dimensional space, where N is the number of attributes. Each side of this hyper-rectangle represents an interval of the domain of its attribute. An Active block contains at least one object in the actual database. The set of Active blocks is indexed using a UB-tree [1], which is a multi-dimensional generalization of B-tree [3] based on Z-order curve [6].

The paper is organized as follows. A short survey about the multi-dimensional indexes and the type of supported queries is presented, then the Osiris system and its main concepts that are necessary to understand the partitioning approach of the object space are explained. The UB-tree indexing and how the object space (called Classification Space) is indexed using UB-trees organization is then presented. Finally, the processing of range queries in our approach is explained.

II. Multi-dimensional indexing

The indexing data structures which index data based on a single dimensional key like binary-tree, B-tree, etc. are efficient in database systems to support operations on data like retrieval, deletion, etc. However, these indexes are not suitable to situations where queries have multiple search keys [15], such as range queries and similarity queries, which play an

S. Housseno Laboratoire TIMC-IMAG, Faculté de Médecine de Grenoble 38706 La Tronche cedex – France fax : 0033 4 76 76 88 44, e-mail: samer.housseno@imag.fr
A. Simonet Laboratoire TIMC-IMAG, Faculté de Médecine de Grenoble 38706 La Tronche cedex – France fax : 0033 4 76 76 88 44, e-mail: ana.simonet@imag.fr
M. Simonet Laboratoire TIMC-IMAG, Faculté de Médecine de Grenoble 38706 La Tronche cedex – France fax : 0033 4 76 76 88 44, e-mail: michel.simonet@imag.fr

important role in many current situations such as Data Warehouses, spatial databases, multimedia databases, computer graphics, Geographical Information Systems (GIS), etc. To deal with these new database systems and applications; the representation of multi-dimensional data is an important issue.

Multi-dimensional data is to be seen as a collection of points (objects) in a higher dimensional space, i.e., whose dimension is greater than 1 [20]. For these object spaces, high-dimensional indexing methods have been considered as an important means to facilitate fast query processing. To support efficient retrieval in such high-dimensional databases, indexes are required to prune the search space.

Multi-dimensional indexes are required to support queries such as [15]:

1) Complete/Partial Range queries, and
2) Similarity queries:
    a) Similarity range queries: « find all objects in the database which are within a given distance from a given object », and
    b) K-nearest neighbor (KNN) queries: « find the K-most similar objects in the database with respect to a given object ».

This paper deals with complete range queries.

- Informally, a complete range query RQ is of the form « Find all objects whose attribute values fall within a certain given range » [15]. For this type of query, a class C with n attributes can be considered as a n-dimensional space E_C, defined as a Cartesian product of the domains D_1 x D_2 x …x D_n, where the dimension D_i represents the domain of an attribute $Attr_i$. In this space, an object o_j, represented by the n-uple :<v_{j1}, v_{j2}, …, v_{jn}>, represents a point in the E_C space and v_{ji} represents its coordinate in the dimension D_i. In this space, a query is defined as: {o ∈ Ec | o ∈ RQ}.
- Formally [15], if δ_i is the range of a query along the dimension D_i. The result of the query: Q={δ_1, δ_2,...,δ_n}; is the collection of {o_j∈ E_C} that satisfy the condition $v_{j1} \in \delta_1$, $v_{j2} \in \delta_2$, … , $v_{jn} \in \delta_n$.

Multi-dimensional indexes such as R-trees [21] are not scalable in terms of the number of dimensions. When the dimensionality of data is high, the performances of R-tree-based index structures deteriorate rapidly [17]. Another type of indexing structures such as Grid files [19] and EXCELL [18] have been proposed. In this type of structure, data partitioning is dynamic, i.e., for each attribute, it is based on the distribution of the attribute values on its domain at a given moment.

To resolve the 'dimensionality curse' [15], [9] in these methods, some authors [5] have proposed to reduce the dimensionality of data by transforming data objects from a multi-dimensional space into one-dimensional space. Space Filling Curve is a way of mapping the multi-dimensional space into one-dimensional space [6]. A space filling curve imposes a linear order on the points by assigning an identifier to each one. A one-dimension index may then be used to index points (objects) by their identifier. As a result, the size of the indexed data is reduced, resulting in a smaller index size and a faster algorithm for search processing.

A space filling curve technique and one-dimension index are not used to index directly the objects. The indexing method presented in this paper uses them to index the disjoint blocks of the partition of the object space. In this approach, the partition of the object space is a semantic partition because it is based on the static analysis of object data model. This semantic partition provides an efficient query optimization because the query handles sets of objects instead of individual objects. In this type of approach, the phenomenon of partitions overlapping which happens with an index like R-tree is avoided. To explain how disjoint blocks are obtained from the object data model, a short presentation of main notions of Osiris is needed.

III. Osiris basic concepts

A full presentation of the Osiris system is not necessary to understand the semantic partition of the object space. The main notions that are useful for this purpose: P-Types, views, attributes and constraints are presented below.

Definitions

P-Types. The global object space is divided into disjoint sub-spaces where each sub-space, called a P-Type space, concerns the objects of a same family. « P » stands for the French « partagé » which means « shared ». As an example, the data model of the Information System of a car insurance company consists of three P-Types: the P-Type Car, the P-Type Client and the P-Type Contract. In this data model, the object o1=(name: Jack, age:40, sex: m, ClientId: 14524784AA, address:'01 Grande rue, Grenoble, FRANCE',…) belongs to the object sub-space of the P-Type Client, the object o2=(CarRegistration: 254 QDE 38, brand: Smart, doors number: 2, year: 2009,…) belongs to the object sub-space of the P-Type Car and the object o3=(contracRef:14587515, InsuranceCoverages: Bodily Injury, BenefitID: 14524784AA, CarRegistration: 254 QDE 38, duration: 25months,…) belongs to the object sub-space of the P-Type Contract. Fig. 1 shows a simple representation of this data model.

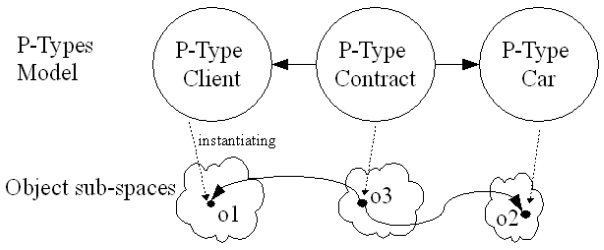

Fig. 1 P-Types of the Car insurance company model

When designing an Information System in Osiris, choosing the P-Types of the application domain depends on the application needs. This is a designer's decision.

P-Types are primitive concepts in the Description Logic paradigm, but not all primitive concepts are P-Types. The objects of a P-Type are meant to be shared by different categories of users, hence through different points of view, which are expressed by views in Osiris.

Views. A P-Type is organized as a hierarchy of views rooted in a minimal view that contains all the objects of the P-Type. A view is defined by the view(s) it specializes (except the minimal view, which is the root of the hierarchy), by its own attributes and by its own constraints defined on attributes, i.e., its own attributes and the inherited attributes from parent views.

Attributes. Attributes are defined within views. An attribute has a name and a type. The type of an attribute can be predefined (INTEGER, REAL, BOOLEAN, CHARACTER, STRING), a P-Type (i.e., a reference to a P-Type), and a collection (set, list) of a predefined type or a P-Type. Although attributes can be defined in any view (possibly in several views) of a P-Type, for the sake of simplicity we will consider in this paper that the attributes of a P-Type are defined in the minimal view and their domain is restricted by constraints in the views that constitute the P-Type.

Constraints. Constraints are Horn clauses whose literals are elementary Domain Predicates (in short DPs), i.e., predicates of the form *Attr* ∈ *Domain*, where Domain can be an interval (e.g., [10, 20]) or a set of enumerated values (e.g., {true, false}, {1, 3, 5, 7}, {blue, red, brown, yellow}).

Example. The P-Type PERSON is shown in Fig. 2 and Fig. 3 with very simple views.

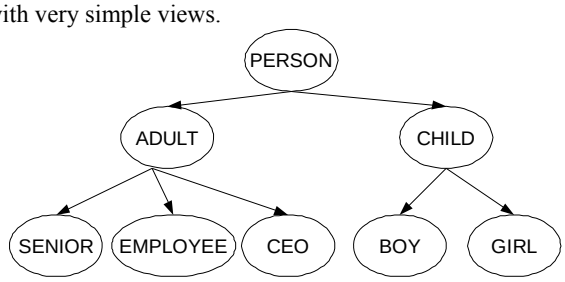

Fig. 2 P-Type PERSON

```
view PERSON          -- Minimal view of the P-Type PERSON
 attr
   name: STRING;
   id: INT;
   sex: CHAR in {m, f}; -- Domain constraint: Sex ∈ {m, f}
   age: INT in [0..140]; -- Domain constraint: Age ∈ [0..140]
   owns: setof CAR;     --The P-Type CAR is defined elsewhere
   salary : INT ≥ 0;    -- Domain constraint: salary ∈ [0..SUP]
   …
   age < 18 ➔ salary < 1200,00
 end PERSON;
```

```
view ADULT: PERSON  -- Specializes the view PERSON
  age ≥ 18              -- Domain constraint: age ∈ [18..140]
  salary ≥ 600,00
end ADULT;
view SENIOR: ADULT      -- Specializes the view ADULT
  age ≥ 65
end SENIOR;
view CHILD: PERSON      -- Specializes the view PERSON
  age < 18              -- Domain constraint: age ∈ [0..18[
end CHILD;
view GIRL: CHILD        -- Specializes the view CHILD
  sex = f               -- Domain constraint: sex ∈ {f}
end GIRL;
view BOY: CHILD         -- Specializes the view CHILD
  sex = m               -- Domain constraint: sex ∈ {m}
end BOY;
view EMPLOYEE: ADULT    -- Specializes the view ADULT
  salary ≥ 1200,00
end EMPLOYEE;
view CEO: ADULT         -- Specializes the view PERSON
  salary > 3000,00
end CEO;
```

Fig. 3 Description of the P-Type PERSON

The views and the P-Type defined above are very simple, in order to support the presentation of the Classification Space, which supports the indexing mechanism that is the basis of the semantic optimization mechanism.

Stable SubDomains. In a P-Type T, for each attribute $Attr_i$ let $P(Attr_i)$ be the set of elementary predicates on $Attr_i$ that appear in all the assertions of all the views of T. Each elementary predicate has the form $Attr_i \in d_{ij}$ where d_{ij} is a subset of the domain of definition of $Attr_i$, i.e., D_i and $j \in [1..NumSBD_i]$ where $NumSBD_i$ is the number of the subset of the domain D_i.

An elementary predicate, i.e., $Attr_i \in d_{ij}$, determines a *partition* of D_i into two elements: d_{ij} and $(Di - d_{ij})$. The product of all the partitions defined by the predicates of $P(Attr_i)$ constitutes a partition of D_i [14]. An element of this partition constitutes a block called Stable SubDomains (SSD), written d_{ij}. A subdomain is *Stable* because it verifies the *stability property* of an object with respect to the related attribute.

Definition 1: *stability property* of an attribute: When the value of an attribute $Attr_i$ of an object O_k varies within a SSD, e.g., d_{ij} ; $j \in [1..NumSBD_i]$, the object O_k continues to satisfy exactly the same predicates of $P(Attr_i)$.

A static analysis of the P-Type description allows determining the list of the SSDs of all the classifying[1] attributes, see Fig. 4. Explaining the static analysis technique is outside the scope of this paper.

[1] Classifying attributes are the attributes that take part in at least one domain constraint in a view of a data model.

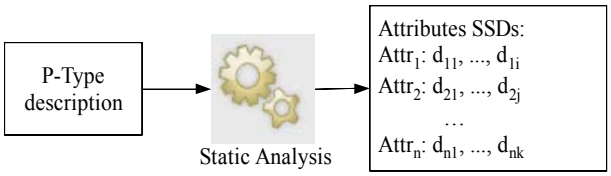

Fig. 4 Static analysis of P-Type description and its output

Given the set of constraints defined in all the views of the P-Type PERSON, the products of the partitions for the attributes *age*, *sex* and *salary* lead to the following partitioning of their domain:

age: d_{11} = [0, 18[, d_{12} = [18, 65[, d_{13} = [65,140]

sex: d_{21} = {f}, d_{22} = {m}

salary: d_{31} = [0, 600[, d_{32} = [600, 1200[, d_{33} = [1200, 3000[, d_{34} = [3000, SUP[

SDS_{ATTR} is the set of all the stable subdomains of the attribute Attr. In the considered example, the SSDs of the attributes are:

SSD_{age} = {d_{11}, d_{12}, d_{13}},

SSD_{sex} = {d_{21}, d_{22}} and

SSD_{salary} = {d_{31}, d_{32} , d_{33}, d_{34}}

Validity of SSD for a view: A stable subdomain is valid for a view v if and only if:

1) It is valid for its parent views and
2) It satisfies its constraints.

Validity of a SSD for a P-Type: A stable subdomain is valid for a P-Type T iff it is valid for the minimal view of T.

Eq-class. The *Classification Space* is a *subset* of the Cartesian product of SSDs of all the classifying attributes of the P-Type:

$SSD_1 \times SSD_2 \times \ldots \times SSD_i \times \ldots \times SSD_n = \{<d_{1i}, d_{2j}, \ldots, d_{nk}> \mid d_{1i} \in SSD_1 \wedge \ldots \wedge d_{nk} \in SSD_n\}$

Where SSD_j represents the set of stable subdomains of the attributes $Attr_j$, for j ∈ [1..N], where N is the number of classifying attributes.

The Classification Space is a N-dimensional space where each element, called *Eq-class* (for **Eq**uivalence **Class**) is a hyperrectangle represented by a N-uple of stable subdomains, i.e., $<d_{1i}, d_{2j}, \ldots, d_{nk}>$. See Fig. 5.

For the graphical representations, we limit ourselves to the 3D space. Thus, considering only the three attributes *age*, *sex* and *salary*, the Classification Space of the P-Type PERSON is represented in Fig. 6.

Validity of an Eq-class for a view: an Eq-class is valid for a view iff all the SSDs of its N-uple are valid for this view.

Validity of an Eq-class for a P-Type: an Eq-class is valid for a P-Type iff all the SSDs of its N-uple are valid for the P-Type.

The valid Eq-classes of a P-Type PERSON are represented in bold on Fig. 6. For example, the Eq-class (d_{13}, d_{22}, d_{34}), that contains among others the object (age=65, sex=m, salary = 4000) is valid because d_{13}, d_{22} and d_{34} are valid, whereas any object of the Eq-class (d_{11}, d_{22}, d_{33}) is invalid, because any person aged less than 18 (age ∈ d_{11}) can only satisfy d_{31}= [0..600[or d_{32}= [600..1200[.

The stability property of an attribute (see Definition 1) can be extended to the whole Classification Space.

Stability property of an Eq-class: all the objects of the same Eq-class have the same validity for all the views of a P-Type.

Corollary: when one or more attribute of an object is modified while remaining in the same Stable SubDomain, the object continues to satisfy the same predicates, hence the same assertions and consequently the same views.

As two objects of the same Eq-class satisfy the same assertions, and consequently validate (or invalidate) the same views, it is possible to determine *a priori* the views that the objects of an Eq-class satisfy. As a consequence, it is possible to associate with each view the set of Eq-classes that validate it.

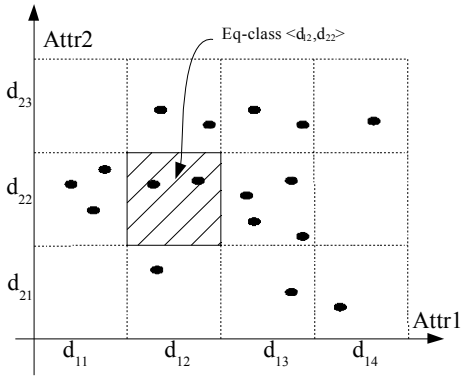

Fig. 5 Partition of the object space into equivalence classes (Eq-classes) in a 2-dimensional space with Eq-class (d_{12},d_{22}) as example

Accordingly, the Classification Space is a partitioning of the object space into equivalence classes such that all the objects of an Eq-class are classified into the same set of views and therefore satisfy the same set of constraints.

An active Eq-class is an Eq-class that contains at least one object.

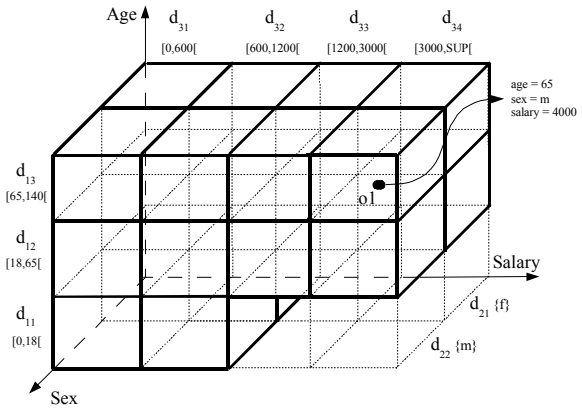

Fig. 6 Classification Space of the P-Type PERSON

IV. Indexing active Eq-classes in Osiris

Before explaining the use of UB-trees in the indexing engine of the Osiris system, UB-trees [1] and the DRU algorithm [10] will be presented in this section.

Single-attribute indexing data structures are well tested and optimized. The need to index on many attributes and the emergence of multi-dimensional applications motivate the adaptation of single-attribute indexing data structures in these contexts. The transformation from multi-dimensional space to uni-dimensional space is an important and necessary step to use single-attribute indexing data structures. UB-tree indexing [1] is inspired by this approach.

UB-trees are a multi-dimensional generalization of B-trees [3] based on the Z-curve space filling curve [6].

A. Space-Filling Curve

The Space-Filling Curve is a method to map a multi-dimensional space into a one-dimensional space. In 1890, G. Peano was the first mathematician who constructed a curve that maps from the unit interval [0,1] to the unit square $[0,1]^2$ [11]. In 1891, Hilbert constructed a mapping of the whole space [12] and many curves have been proposed since [6].

Each curve has its own mapping function: Z order, Peano curve, Hilbert order, Gray order, U order, etc. Each curve visits the points of the multi-dimensional space one after another. The main difference between the curves is the choice of the next point to be visited. The multi-dimensional data universe is linearized to a one-dimensional space by representing a multi-dimensional point by its position on the curve. Consequently, the points are ordered, which permits to index them using a single-attribute indexing data structure, e.g., UB-trees. UB-trees are based on the Z-curve, which is presented in the next section. For other curves, see [6].

Space-Filling Z-curve

The mapping of a point from multi-dimensional space into one-dimensional space is done by calculating its position on the Z-curve, which is called its *Z-value*. Based on the binary representation, the Z-value is assembled by cyclically taking a bit from each coordinate of a point and appending it to those taken previously. Fig. 7

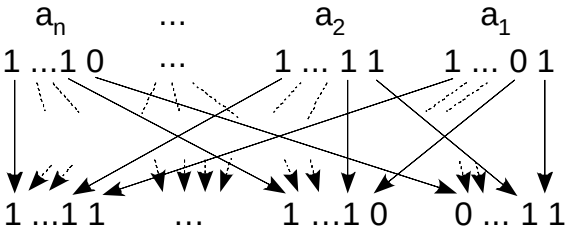

Fig. 7 Bit-interleaving algorithm in N-dimensional space

Fig. 8 shows how the Z-curve fills the two dimensional space.

The cost of Z-value construction is cheap and the work of [22] demonstrates that it has very good characteristics.

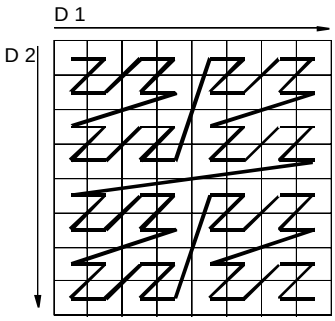

Fig. 8 Space-Filling Z-curve in bi-dimensional space

B. UB-tree

A UB-tree is a balanced multi-dimensional data structure based on the space filling Z-curve [6] and B-trees [3].

In a UB-tree, a Z-value, which is a position of a point on the Z-curve, is called a Z-address. Z-regions represent clusters of points in the indexed space. A Z-region is bounded by two Z-addresses which are the lower and the upper Z-addresses inside it. The Bounding UB-tree (BUB-tree) [4] does not index the Z-regions which do not contain objects (the dead space).

The UB-tree offers a hierarchical representation of space and also it partitions the whole space into a set of disjunctive but consecutive Z-regions (Z-intervals). Each Z-region containing the indexed data is inserted into one leaf node in the UB-tree. On the other hand, the inner nodes contain super-Z-regions [10]. A super-Z-region bounds all the super-Z-regions in its subtrees.

The algorithms for insertion, deletion and point queries are similar to those implemented in B-trees except that the Z-address of the manipulated data must be computed before the execution of an algorithm. Due to the nature of range queries and the mapping into one-dimensional space, this query has its own algorithm in the UB-tree.

Range query processing

For range queries, the linear DRU algorithm proposed in [10] is used because its performance is better than the original linear algorithm proposed by Bayer-Markl [1], [2].

This algorithm is based on the intersection operation between the range query and the (super-)Z-regions mapped in the inner and leaf nodes. If a super-Z-region intersects the range query, so do its children. For more details, the reader is referred to [10].

The complexity of this algorithm is linear according to Z-address bit-length; i.e., O(n log(|D|)). A path stack is used to keep the current path being processed. The main steps of the DRU algorithm are:

1) Compute the Z-address of the query box lower and higher bounds, (Z_{lb} and Z_{hb} respectively)
2) Find the Z-region (leaf) which contains the Z_{lb}, set it as the current leaf and push its path onto the stack.
3) Search in the current leaf for data tuples that satisfy the range query.

4) If the lower bound of the right-neighbour-leaf Z-region is inside the range query, set it as the current leaf and goto 3.
5) The top of the stack is popped. It contains the parent node (node P) of the last treated leaf.
6) Peek node P to find an entry pointing to the next query-intersected node (node R). We have two cases:
    a. No such entry is found: remove node P from the stack and repeat step 6.
    b. One entry is found: retrieve the node R and push it onto the stack. If node R is a leaf, then goto step 3 otherwise repeat step 6.

In a two-dimensional space, Fig. 9 shows the super Z-regions and the Z-regions represented in the UB-tree. The tree itself was not represented for the sake of clarity. At each level, regions are larger but less numerous than those at the immediately lower level. The treatment of the RQ starts at the root level, where two super-Z-regions intersect RQ.

In the next level (level 1), three smaller super-Z-regions intersect the RQ. At level 2, four smaller super-Z-regions intersect the RQ. In the last level, which is the leaf level, four Z-regions intersect the RQ.

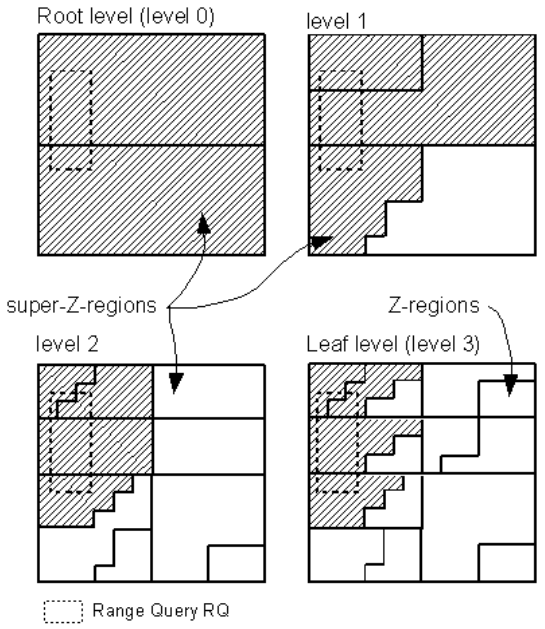

Fig. 9 Common regions between a range query RQ and (super)-Z-regions at each level in the UB-Tree

C. Using UB-trees in the Osiris indexing engine

In the indexing method, one UB-tree is used to index the active Eq-classes by their identifiers, instead of indexing directly the objects. This tree is called Active Eq-classes UB-tree (*AEC UB-tree*). The objects of each Eq-class are also indexed by another UB-tree, which is called Active Eq-class k Objects UB-tree (*AECk-O UB-tree*); k is the Eq-class identifier. A Z-region in an AEC UB-tree contains a set of Eq-class identifiers and pointers to the appropriate AECk-O UB-trees. A Z-region in an AECk-O UB-tree is a set of indexed objects (Fig. 10).

Recent systems have a large volume of RAM. Since the AEC UB-tree Z-regions contain a set of Eq-class identifiers and pointers, a whole AEC UB-tree and possibly the inner nodes of AECk-O UB-trees can be stored in the RAM. This is an efficient organization in the case of very large volumes of data.

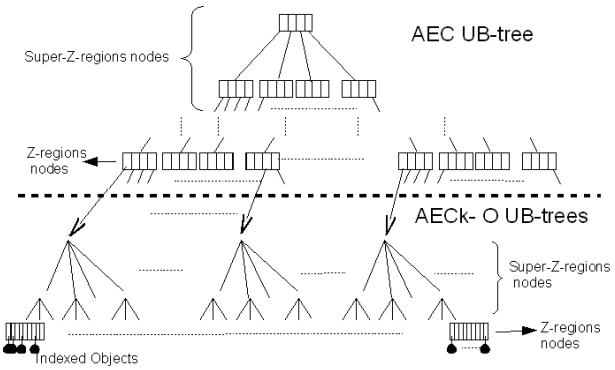

Fig. 10 AEC UB-tree and AECk-O UB-trees organization

V. Queries in Osiris

For the insertion, deletion, and point query algorithms, firstly, the Z-address of the Eq-class of the object is calculated and then the UB-tree original algorithm is called. This paper deals complete range queries.

A. Range queries processing in Osiris

A query $Q=\{\delta_1,\ldots, \delta_j,\ldots,\delta_n\}$ such that $l_1\leq\delta_1\leq h_1,\ldots, l_j\leq\delta_j\leq h_j \ldots, l_n\leq\delta_n\leq h_n$ can be seen as a hyper-rectangle in the N-dimensional space. This hyper-rectangle is bounded by a lower bound point P_l and an upper bound point P_u such that $P_l=(l_1,l_2,\ldots,l_n)$ and $P_u=(u_1,u_2,\ldots,u_n)$. In Osiris, these two points are transformed into the Z-addresses of Eq-classes.

To illustrate how UB-trees are used in Osiris, a two-dimensional space is used, with the dimensions *age* and *salary* of the P-Type PERSON (Fig. 11). Considering these two dimensions, the Eq-classes (d_{11}, d_{33}) and (d_{11}, d_{34}) are excluded from the P-Type, because of the constraint `age<18 ➔ salary <1200`. The set of possibly valid Eq-classes is surrounded by bold lines.

In a N-dimensional space, an Eq-class is designated by a N-uple of SSDs. In the example shown in Fig. 11, the Eq-classes are written (d_{11}, d_{31}), (d_{11}, d_{32}), …, (d_{13}, d_{34}). This is a bi-dimensional representation. To obtain a one-dimensional representation, each SSD is assigned a binary code that is unique for each attribute. For example:

age $d_{11} = 00$, $d_{12} = 01$, $d_{13} = 10$
salary $d_{31} = 00$, $d_{32} = 01$, $d_{33} = 10$, $d_{34} = 11$

Applying the Z-order or (bit-interleaving), the unique

identifiers associated with the Eq-classes are computed. They are called Eq-class z-addresses:

$(d_{11}, d_{31}) = 0000$, $(d_{11}, d_{32}) = 0001$

$(d_{12}, d_{31}) = 0010$, $(d_{12}, d_{32}) = 0011$, $(d_{12}, d_{33}) = 0110$, $(d_{12}, d_{34}) = 0111$

$(d_{13}, d_{31}) = 1000$, $(d_{13}, d_{32}) = 1001$, $(d_{13}, d_{33}) = 1100$, $(d_{13}, d_{34}) = 1101$

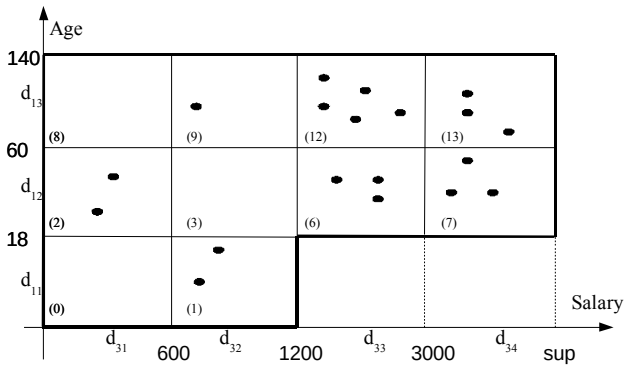

Fig. 11. Eq-classes of the P-Type person

The size of the binary code of an attribute is determined by the number of SSDs of this attribute. The decimal numbering corresponding to the binary code of the z-addresses of Eq-classes is presented in the lower left corner of each Eq-class in Fig. 11.

In the example shown in Fig. 11, the Eq-classes (0), (3) and (8) will not be represented since they do not contain any object (non active Eq-classes).

For RQ {25 ≤ age ≤ 62 and 500 ≤ salary ≤ 1500} (Fig. 12), the lower and the upper Eq-class Z-addresses for RQ are respectively 2 and 12. Since the active Eq-classes are 2, 6, 9 and 12, the result of processing the DRU algorithm on the AEC UB-tree is 2, 6, 9 and 12. These are not the objects which satisfy RQ. They are the Eq-class Z-addresses which may contain objects satisfying the query. Another step is necessary to search the appropriate AECk-O UB-trees for objects lying inside RQ.

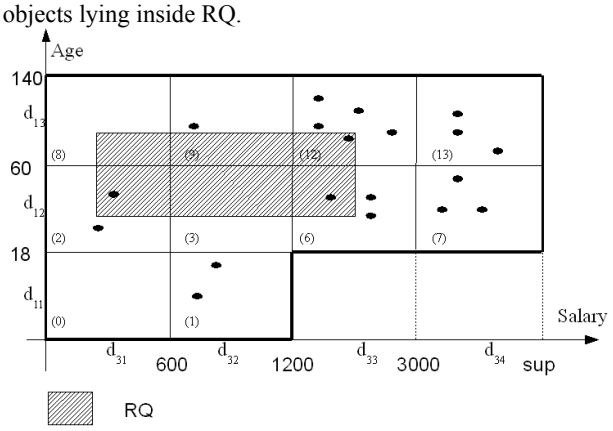

Fig. 12 Eq-classes for RQ

VI. Conclusion

The indexing method presented in this paper is designed for the P-Type object data model that aims at unifying databases and knowledge bases [13]. This model has been implemented in the KB-DBMS prototype Osiris. As a DBMS, it is based on views defined by the views they specialize, their own attributes and logical constraints on attributes. A static analysis of the object data model enables the system to partition the object space into a so-called Classification Space, whose elements are no longer individual objects but Equivalence Classes, named Eq-classes. The Classification space is used to optimize integrity checking, object classification, and for primary object indexing.

Moreover, in order to provide an efficient access to the objects referenced by an Eq-class, the UB-tree structure is used, which is a multi-dimensional generalization of B-trees based on the Z-curve space filling curve.

Our approach can be applied to any relational system by taking into account the integrity constraints of the database as a basis for the determination of SSDs and Eq-classes.

The indexing method presented in this paper is an efficient organization in the case of very huge volume of data. The processing of the DRU algorithm on the AEC UB-tree eliminates an important number of objects which are not inside the range query.

At present, the performance of the indexing method based on UB-trees is compared and analyzed with the performance of different indexing methods used in DBMSs.

VII. References

[1] Bayer, R.: The universal B-Tree for multi-dimensional Indexing: General Concepts. In World-Wide Computing and its Applications ’97 (WWCA’97), Lecture Notes on Computer Science. Springer Verlag, Tsukuba, Japan (1997)

[2] Markl, V.: Processing Relational Queries using a Multidimensional Access Technique. PhD thesis, DISDBIS, Band 59, Infix Verlag (1999)

[3] Bayer, R., McCreight, E.: Organization and Maintenance of large ordered Indexes. In Acta Informatica 1, pp. 173--189 (1972)

[4] Ramsak , F.: The BUB-Tree. In Proceedings of 28rd VLDB International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, China (2002)

[5] Orenstein, J.A., Merrett, T.H.: A Class of Data Structures for Associative Searching. In Proceedings of the Third ACM SIGACT-SIGMOD Symposium on PODS, April 2-4, pp. 181--190. ACM (1984)

[6] Sagan, H.: Space-Filling Curves, Springer; 1 edition (September 2, 1994), ISBN-10: 0387942653, ISBN-13: 978-0387942650

[7] Otoo, E. J.: A Mapping Function for the Directory of a Multidimensional Extendible Hashing, Proceedings of the 10th International Conference on Very Large Data Bases(VLDB), pp. 493–-506, San Francisco, CA, USA (1984)

[8] Berchtold, S., Keim, D. A.., Kriegel, H.: The X-tree: An Index Structure for High-Dimensional Data, Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB), pp. 28—39, India (1996)

[9] Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: index structures for improving the

performance of multimedia databases, ACM Comput. Surv. 33 (3), pp. 322–373, (2001)
[10] Skopal, T., Krátký, M., Pokorný, J., Snášel, V.: A new range query algorithm for universal B-trees, Information Systems, Vol. 31, Issue 6, pp. 489 – 511, (September 2006)
[11] Peano, G.: Sur une courbe qui remplit toute une aire plaine, Mathematishe Annalen, 36, pp. 157—160, (1890)
[12] Hilbert, D.: Ueber die stetige Abbildung einer Line auf ein Flächenstück, Mathematische Annalen, 38: pp. 459–460, (1891)
[13] Simonet, A., Simonet, M.: OSIRIS : an Object-Oriented system Unifying Databases and Knowledge bases, KBKS'95 : Towards Very Large Knowledge Bases, Enschede, The Netherlands, pp. 217--227, N. Mars Ed., IOS Press, (1995)
[14] Stanat, D., McAllister, D. : Discrete Mathematics in Computer Science, Prentice Hall (1977)
[15] Yu, C.: High-Dimensional Indexing: Transformational Approaches to High-Dimensional Range and Similarity Searches, Springer (2002)
[16] Simonet, A., Simonet, M.: Objects with Views and Constraints : from Databases to Knowledge Bases, Object-Oriented Information Systems OOIS'94, Springer Verlag, pp. 182--197, London (1994)
[17] Bertino, E., Chin, O.B., Sacks-Davis, R., Tan, K., Zobel, J., Shidlovsky, B., Andronico, D.: Indexing Techniques for Advanced Database Systems. Kluwer Academic (1997)
[18] Tamminen, M., Sulonen, R.: The excell method for efficient geometric access to data, Annual ACM IEEE Design Automation Conference, Proceedings of the 19th conference on Design automation, pp. 345--351 (1982)
[19] Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The Grid File: An Adaptable, Symmetric Multikey File Structure, ACM Trans. Database Syst, vol. 9(1), pp. 38--71 (1984)
[20] Samet, H.: Foundations of Multidimensional and Metric Data Structures, ISBN-10: 0123694469, ISBN-13: 978-0123694461, Morgan Kaufmann (2006)
[21] Guttman, A. : R-trees: a dynamic index structure for spatial searching, Proceedings of the 1984 ACM SIGMOD international conference on Management of data, SIGMOD 84, pp. 47—57, Boston, Massachusetts (1984)
[22] Mokbel, M. F., Aref, W. G., Kamel, I.: Performance of multi-dimensional space-filling curves, Geographic Information Systems, Proceedings of the 10th ACM international symposium on Advances in geographic information systems, pp.149 – 154 McLean, Virginia, USA (2002)