Supporting RFID-based Item Tracking Applications in Oracle DBMS Using a Bitmap Datatype

Ying Hu Seema Sundara Timothy Chorma Jagannathan Srinivasan

Oracle
One Oracle Drive
Nashua, NH 03062
USA
(ying.hu, seema.sundara, timothy.chorma, jagannathan.srinivasan}@oracle.com

Abstract

Radio Frequency Identification (RFID) based item-level tracking holds the promise of revolutionizing supply-chain, retail store, and asset management applications. However, the high volume of data generated by item-level tracking poses challenges to the applications as well as to backend databases. This paper addresses the problem of efficiently modeling identifier collections occurring in RFID-based item-tracking applications and databases. Specifically, 1) a bitmap datatype is introduced to compactly represent a collection of identifiers, and 2) a set of bitmap access and manipulation routines is provided. The proposed bitmap datatype can model a collection of generic identifiers, including 64-bit, 96-bit, and 256-bit Electronic Product Codes™ (EPCs), and it can be used to represent both transient and persistent identifier collections. Persistent identifier collections can be stored in a table as a column of bitmap datatype. An efficient primary B+-tree–based storage scheme is proposed for such columns. The bitmap datatype can be easily implemented by leveraging the DBMS bitmap index implementation, which typically manages bitmaps of table row identifiers. This paper presents the bitmap datatype and related functionality, illustrates its usage in supporting RFID-based item-tracking applications, describes its prototype implementation in Oracle DBMS, and gives a performance study that characterizes the benefits of the bitmap datatype.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment

Proceedings of the 31st VLDB Conference,
Trondheim, Norway, 2005

1. Introduction

Radio Frequency Identification [14] is emerging as key technology for a wide-range of applications, including supply chain, retail store, and asset management. RFID tags can be associated with objects, such as pallets, cases, and even individual items. By placing RFID tag readers at various locations, one can track the movement of objects through supply chain networks, namely, from manufacturers to retailers, then to consumers. Such item-level tracking can greatly enhance the efficiency of business operations. However, it can also result in data explosion, and hence, efficient means are needed to model such data.

This paper addresses this problem by proposing a bitmap datatype to efficiently model different collections of identifiers that typically occur in item-tracking applications. A key observation is that although individual items need to be tracked, they can be tracked more efficiently by tracking the groups to which they belong. For instance, a group could be made up of items in the same proximity (e.g., on the same shelf, or in the same shipment). Similarly, items could belong to a group based on a shared property (e.g., items of an identical product, or manufacturer, or items with the same expiration date).

One alternative to tracking a group of identifiers is to maintain an item count for the group, where the count can be viewed as a transformation obtained from an identifier collection. However, there is a loss of information during the transformation that makes the approach unsuitable for the RFID domain, especially when individual items need to be tracked. For example, in a product recall application, the exact set of items needs to be identified, as opposed to

the count. For tracking such collections, the paper proposes the following:

- a bitmap datatype that compactly[1] represents a collection of RFIDs, that is, their EPCs, and
- the use of bitmap operations to access and manipulate such collections. These include conversion operations (EPCs to bitmap, bitmap to count, bitmap to EPCs), logical operations (AND, OR, MINUS, XOR), membership testing, bitmap maintenance, and comparison operations.

By using the bitmap datatype, and its associated access and manipulation routines, efficient item-tracking applications can be built.

We propose the following scheme for inventory tracking. Inventories are maintained at periodic checkpoints using bitmaps. A bitmap datatype instance is used to represent the items present in a particular group, at each checkpoint. A group could consist of a single product, or a collection of products that share a property.

A Motivating Example: Consider a product recall application, where products need to be recalled from various retail stores because of a defect. Consider a table that maintains the product inventory for retail stores, per our inventory tracking scheme. In the absence of the bitmap datatype, a list of EPCs needs to be maintained. The inventory for a product at various retail stores can be represented using a collection type as shown in Table 1.

Table 1: Product_Inventory with EPC Collections

product_id	store_id	time	item_collection
p1	s1	t1	epc11
			epc12
			…
p1	s1	t2	epc21
			epc22
			…
…	…	…	…
p1	s2	t1	epck1
			epck2
			…
…	…	…	…

Alternately, the EPC list can be represented using a bitmap datatype as shown in Table 2, which can lead to significant storage savings, as we will show.

Table 2: Product_ Inventory with EPC Bitmaps

product_id	store_id	time	item_bmap
p1	s1	t1	bmap1
p1	s1	t2	bmap2
…	…	…	…
p1	s2	t1	bmapk
…	…	…	…

Furthermore, bitmap operations can be used to perform interesting queries. For example, the following query can be used *to identify the stores that currently have recalled items*:

```
SELECT store_id
FROM Product_Inventory
WHERE bmap2Count(bmapAnd(item_bmap,
      epc2Bmap(<recall_items>))) > 0
      AND product_id = <recall_product_id>
      AND time = <current_date>;
```

Thus, a bitmap AND is sufficient to identify the recalled items from the shelves.

The key benefits of introducing a bitmap datatype are:

- It compactly represents a collection of identifiers and the transformation is loss-less, i.e., one can always extract individual identifiers belonging to the collection, and
- The typical operations on a pair of collections can simply be performed using bitmap operations on the corresponding bitmaps.

The bitmap datatype can be used efficiently to model both transient and persistent identifier collections. The persistent identifier collections can be stored in a column of bitmap datatype. For such bitmap columns, a primary B+-tree based storage scheme is proposed, which is compact in storage and provides efficient access to the internal bitmap structure.

We have implemented a prototype of the bitmap datatype in Oracle DBMS using the database's type extensibility mechanism. The implementation reuses much of the underlying database bitmap index code, which creates and manipulates compressed bitmaps for row identifier collections. Performance experiments conducted with an example data set, using collections of sizes ranging from 10 to 100,000, implemented as compressed bitmaps, show storage savings on the order of 2 to 8 times over native collections. The query performance on collections modelled as bitmaps is the same or better than collections stored natively. The bitmap datatype based configuration shows performance gains for queries on collections of size > 1000. These gains are in the range of 10% to 1280%.

Frequent incremental maintenance on bitmap datatype columns is costly. Hence, the bitmap datatype is more appropriate for scenarios requiring initial bulk-load followed by batch updates at periodic intervals. For applications requiring frequent incremental maintenance, the following hybrid variation of the inventory tracking scheme is proposed. The inventories are maintained at periodic checkpoints using bitmaps. For changes occurring between checkpoints, an item-level table is maintained. Queries for the specified time period are answered by merging the latest checkpoint bitmap with the corresponding duration's item-level data. Section 3.4 briefly describes this hybrid inventory-tracking scheme.

Supporting the bitmap datatype for modelling EPC collections would allow developing efficient item-tracking applications. Also, this functionality opens up the possibility of building interesting RFID-based analytics for comprehensive supply-chain management.

The bitmap datatype concept is generic and can be used in any large-scale application. However, our focus is

[1] The bitmap itself is compressed using the byte aligned data compression scheme [4].

on the bitmap representation of EPCs and its potential use in addressing the challenges faced in RFID applications.

The key contributions of the paper are:

- A bitmap datatype in a DBMS, which can be used to model both transient as well as persistent identifier collections.
- The use of a bitmap datatype and associated operations to support RFID-based item tracking applications.
- An extensible scheme for the bitmap datatype, which allows creation of bitmaps for classes of identifiers. Currently, the bitmap datatype can be used to represent 64-bit, 96-bit, and 256-bit EPC collections.
- An efficient primary B+-tree based storage and access mechanism for a bitmap datatype column.
- An implementation prototype of the bitmap datatype in Oracle DBMS, using underlying bitmap index code for table row identifiers, and its performance characterization.

1.1 Related Work

There are quite a few papers describing bitmap indexes, their usage, and performance characteristics [4, 5, 6, 7, 8, 12]. Also, several database systems including IBM DB2, Oracle, and Sybase support bitmap indexes. However, none of the systems support a bitmap datatype. Worth noting is MySQL [13], which has a notion of a set type. However, the set type is used to represent enumerated sets of small sizes (64 values). It does not address the issue of modelling a set or collection of identifiers belonging to a large sequence, as a bitmap.

Both C++ and Java programming languages support a bitset class, which can be used to represent a set of bits. A set of member functions and operations are also supported on this class. Unlike the bitmap datatype, which can be used to represent both transient and persistent object collections, the bitset class is meant for representing only transient objects. Also, the bitset class does not address the issue of compactly representing a bitset object by compressing the sequences of 1s and 0s, and supporting operations on the compressed objects.

With regard to RFID-based item tracking applications, interesting usage scenarios have been considered ([9, 10, 11]). However, none of these applications have considered using a bitmap datatype.

1.2 Outline of Paper

Section 2 describes the bitmap datatype and associated functionality. Section 3 illustrates the usefulness of the bitmap datatype by considering several applications. Section 4 presents a prototype implementation in Oracle DBMS. Section 5 presents performance experiments and Section 6 gives conclusion and some future work.

2. Key Concepts

This section gives an overview of EPCs and discusses the bitmap datatype and related functionality.

2.1 EPC Types

The Electronic Product Code™ (EPC) [1, 2, 3] is the next generation of product identification scheme. The EPC is a simple naming scheme that uniquely identifies objects (items, cases, pallets, locations, etc.) in the supply chain. The EPC can be used to express a wide variety of different, existing numbering systems, like the EAN.UCC System Keys, Unique Identifiers, Vehicle Identification Number, and other numbering systems.

Like many current numbering schemes used in commerce, the EPC is divided into numbers that identify the manufacturer and product type. But, the EPC uses an extra set of digits, a serial number, to identify unique items within each group. An EPC number primarily contains:

1. Header, which identifies the length, type, structure, version and generation of EPC.
2. Manager Number, which identifies an organizational entity. The organization is responsible for maintaining unique numbers in the next two fields.
3. Object Class, which identifies a “class”, or type of thing.
4. Serial Number, which is the specific instance of the Object Class being tagged. The managing entity is responsible for assigning unique, non-repeating serial numbers for every instance within each object class.

For our discussions in this paper, we refer to the first 3 fields (header, manager number and object class) collectively as the *epc_prefix*, since they identify a given product by a given manufacturer. The serial number portion of EPC is referred to as *epc_suffix* since it identifies the unique instance of an item within its product family.

2.2 Bitmap Datatype

A new *epc_bitmap_segment* type is defined to represent a collection of electronic product codes (EPCs), which share a common *epc_prefix*.

```
CREATE TYPE epc_bitmap_segment
  (    epc_length          NUMBER,
       epc_suffix_length   NUMBER,
       epc_prefix          RAW,
       epc_suffix_start    RAW,
       epc_suffix_end      RAW,
       epc_suffix_bitmap   RAW
  );
```

The *epc_length* defines the total number of bits in the EPC, *epc_suffix_length* defines the number of bits in the EPC suffix, *epc_prefix* is a raw that holds the common EPC prefix part, (header, manager number and object class), *epc_suffix_start* is a raw that holds the minimal EPC suffix in the epc_bitmap_segment, *epc_suffix_end* is a raw that holds the maximal EPC suffix in the

epc_bitmap_segment and *epc_suffix_bitmap* is a raw that holds the compressed representation of bitmap corresponding to the EPC suffixes (similar to the row identifiers' compressed bitmap as described in [4,5]), all of which are presumed to have a common EPC prefix.

This type can be used to represent different classes of EPCs (i.e., 64 bit, 96 bit and 256 bit). Since the individual attributes within the epc_bitmap_segment type capture EPC type specification information (EPC length, EPC prefix length, EPC suffix length etc.), the epc_bitmap_segment type can represent each of these EPC classes. Furthermore, even within a given EPC class, there are a number of different EPC formats (e.g., Global Location Number-64, Global returnable Asset Identifier-64 etc.), which vary in prefix/suffix length, etc. By breaking up the EPC specification into epc_prefix, epc_suffix_start and epc_suffix_end, these varying formats within the same EPC class can also be easily stored, and distinguished from one another, by the epc_bitmap_segment type.

The epc_bitmap_segment type defined above is used to model a collection of EPCs that occur close together, i.e., they share the EPC prefix and their suffixes are close together. Thus, the epc_bitmap_segment represents a bitmap segment. To model a collection of EPCs that can potentially be arbitrarily dispersed, a multiset type, *epc_bitmap* is defined that can hold collections of bitmap segments (see Figure 1).

```
epc_bitmap: multiset of epc_bitmap_segment type
```

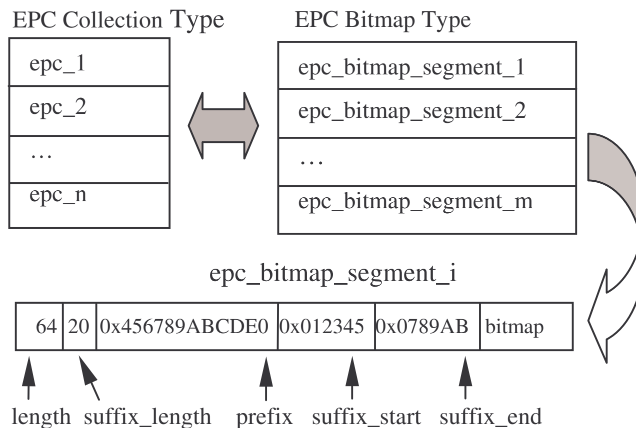

Figure 1: EPC Collection Datatype and Bitmap Datatype (normally, n >> m)

With the epc_bitmap datatype[2] defined, it can now be used either as a transient type in applications or as a persistent type in the database by defining a column of epc_bitmap that can hold multiset values (i.e. nested table type in Oracle).

For operations that take in two epc_bitmaps, their epc_length is first compared to make sure that they belong to the same class (i.e. 64 bit, 96 bit etc.) Then, to further ascertain that they are also defined using the same format, their prefix lengths are compared. Operations across EPC classes, or even across different EPC formats in the same class are not supported.

2.3 Operations on Bitmap Datatype

A set of commonly used bitmap operations is defined for the epc_bitmap. The set of operations can be broadly classified into the following categories:

- **Conversion Operations:** The *epc2Bmap* and *bmap2Epc* functions allow conversion of a set of EPCs into an epc_bitmap and vice versa. They are useful during storage of a set of EPCs in the database as a set of epc_bitmap_segments, and during the subsequent extraction of the EPCs from the database. The *bmap2Count* operation returns a count of the bits that are set in a given epc_bitmap. It can be used to determine the total number of items present in an EPC collection at a given time.
- **Pairwise Logical Operations:** These consist of the *bmapAnd*, *bmapOr*, *bmapMinus*, *bmapXor* functions. Given the shelf level analytics application in Section 3, these operations help in determining the changes on a shelf between two time instances i.e. what items are still on the shelf between time t1 and t2 (bmapAnd), and what items have been added/removed from the shelf between time t1 and t2 (bmapMinus).
- **Maintenance Operations:** Since the epc_bitmap that is being defined can occur as a persistent column in a database table, the column values need to be incrementally maintained whenever a DML is performed on the base table. The *bmapInsert* and *bmapDelete* functions take in an epc_bitmap and a given set of EPCs and construct a new EPC bitmap by inserting/deleting the given set of EPCs into/from the original epc_bitmap.
- **Membership Testing Operation:** The *bmapExists* function can be used to detect the presence/absence of a specific item in a given epc_bitmap.
- **Comparison Operation:** A *bmapEqual* function is defined to compare two epc_bitmaps for equality. An equality check on the epc_bitmaps generated at the distribution center and the receiving warehouse ensures that all the items that were shipped are received.

3. Applications

This section discusses some RFID applications that can benefit from the proposed epc_bitmap. By default, the epc_bitmap is suitable for applications that maintain summary information at periodic intervals. With this assumption, the use of epc_bitmap in certain application scenarios is discussed below. These applications can be broadly categorized as follows:

[2] For the purposes of brevity, epc_bitmap datatype is referred as epc_bitmap later.

- *Retail Store Management*: Shelf Analytics [9], Theft Detection [10], and managing Item Returns.
- *Supply Chain Management:* Managing the recall of defective products [11].
- *Asset Management*: Managing and tracking information for a collection of assets.

Some of these applications may need frequent incremental maintenance. For such applications, a hybrid scheme can be used that employs the epc_bitmap in conjunction with a traditional item-level tracking table. This scheme is discussed in Section 3.4.

Another interesting application of epc_bitmap is to build auxiliary structures that can speed up queries involving epc_bitmap column predicates. This is discussed in Section 3.5.

3.1 Retail Store Management

The examples in this section take into account how RFID-tagged items are managed in a retail store.

3.1.1 Shelf Analytics

A retail storeowner is interested in tracking the items on the store shelves to monitor the purchase, theft and stock maintenance of items.

Consider a table that holds inventory for a store as a collection of shelf-specific item information, as shown in Table 3. We further assume that a RFID reader on the shelf sends an inventory of all the items present on the shelf at regular intervals. The EPCs of all the items present is then converted into an epc_bitmap and stored in our representative table.

Table 3: Shelf_Inventory

shelf_id	Time	item_bmap
sid1	t1	bmap1
sid1	t2	bmap2
…	…	…
sid2	t1	bmapk
…	…	…

To determine *the items added to a given shelf between time t1 and t2*, one could issue the following query:

```
SELECT bmap2Epc(bmapMinus(s2.item_bmap,
                          s1.item_bmap))
FROM Shelf_Inventory s1, Shelf_Inventory s2
WHERE s1.shelf_id = <sid1> AND
      s1.shelf_id = s2.shelf_id AND
      s1.time = <t1> AND s2.time = <t2>;
```

Similarly, reversing the order of arguments to the `bmapMinus` function, one can determine the number of items removed from the shelf. In addition, one can track the count of items by using the `bmap2Count` function. To determine *the count of items on a shelf at a specific point in time*, one could issue the following query:

```
SELECT bmap2Count(item_bmap)
FROM Shelf_Inventory
WHERE shelf_id = <sid1> AND time = <t1>;
```

We can further extend the store model to include RFID readers in the stock room, cash registers etc. Now, if an item is missing, we have a record of where the stolen item was at all times before it was stolen, which can help in determining the culprit. For example, to determine *on which shelf and what time a particular item (say epc1) existed*, the following query can be issued:

```
SELECT shelf_id, time
FROM Shelf_Inventory
WHERE bmapExists(item_bmap, <epc1>);
```

3.1.2 Item Returns

Consider the problem of item returns, which retail stores and consumers face today, specifically without receipts. Even with a receipt, with the lack of a mechanism to uniquely identify an item, retail stores can never be sure that the item being returned was actually purchased from the store it is being returned to.

With the usage of epc_bitmaps, this problem can easily be addressed. Consider a retail store that keeps department inventory as shown in Table 4.

Also, assume that the store has a return policy that allows items to be returned within a period of 90 days from when it was purchased. Now, when an item is brought back to the store for return, a query can be issued against the department inventory table to see if the EPC of the item to be returned (`epc1`) was ever present in the store during the last 90 days.

```
SELECT 1 FROM Dept_Inventory
WHERE bmapExists(item_bmap, <epc1>)
AND dept_id = <did>
AND time > (<current_date> - 90);
```

If the item is not found, it implies that either the item was bought more than 90 days ago and thus is no longer eligible for a return, or the item never existed in the store and wasn't bought from it.

Table 4: Dept_Inventory

dept_id	time	item_bmap
did1	t1	bmap1
did1	t2	bmap2
…	…	…
did2	t1	bmapk
…	…	…

This can not only reduce the return fraud faced by retail stores, but also benefit the customer who genuinely wants to return purchases but has lost the receipt. Since the retail stores can do the lookup based on EPCs, a receipt is no longer the only proof of purchase.

3.2 Supply Chain Management

The epc_bitmap can be used for supply chain management, which typically involves tracking items across manufacturers, distribution warehouses, and retail stores. RFIDs, with their ability to uniquely identify a particular item, are also being considered as a viable solution for product recalls [11].

A key factor during product recalls is the need to determine quickly whether products should be recalled, then rapidly identify recalled products and remove them

from the field. To gain some idea of the potential benefits, imagine a large organization forced to launch a mass recall due to product contamination or tampering. The capability to automatically identify which batch each individual item came from would minimize the cost of withdrawing the product and of conducting the recall, and could potentially allow companies to be far more precise in the withdrawal process.

Storing a set of EPCs as an epc_bitmap allows efficient operation on the sets. As "A Motivating Example" illustrates in section 1, the epc_bitmap and its operations can be very useful to find out which stores currently have recalled items.

3.3 Asset Management

RFID technology is also suitable for tracking assets (such as confidential documents, laptops, containers of hazardous material, equipments, and valuables) within an organization. In addition, they can be used for building rental item applications such as for libraries and video stores.

For example, one can also maintain epc_bitmaps corresponding to collection of items that share a certain property. Let an additional table be maintained that keeps track of a property pertaining to book collections as shown in Table 5.

Table 5: Property_Inventory

Fiction	NonFiction	Adventure	Romance	…
bmap1	bmap2	bmap3	bmap4	…

To determine *the shelves where the books with property Adventure and Romance, are currently present in the library*, one can issue the following query:

```
SELECT s.shelf_id
FROM Shelf_Inventory s
WHERE bmap2Count(
      bmapAnd(s.item_bmap,
              SELECT bmapAnd(p.Adventure,
                             p.Romance)
              FROM Propery_Inventory p)) > 0;
      AND s.time=<current_date>;
```

3.4 Hybrid Scheme for Item Tracking

In some RFID applications, individual item-level information can be maintained using an item-level table. For example, the following table can be used to keep track of items being inserted ('I') or deleted ('D') from a shelf.

Table 6: Shelf Inventory Item Events

shelf_id	time	item	operation
sid1	t1	epc1	'I'
sid1	t2	epc2	'D'
…	…	…	…
sid2	t1	epck	…
…	…	…	…

Although this table allows for updates to be done very frequently, based on the events (insertion and deletion of items), it is not suitable for performing queries that require consolidation of information. For example, to determine the items present on a shelf at a given time, a table scan is required that consolidates the information by scanning over all rows holding events for the shelf. For such queries, pre-computing items present on the shelf at periodic checkpoints and maintaining them in an epc_bitmap-based table (as in Table 3) would be appropriate. When the current data needs to be archived, applications can choose to generate epc_bitmap based information with a finer (or coarser) granularity, or an automatic tuning algorithm can generate the epc_bitmaps, optimally based on the query time and data storage and other variants for the applications.

Thus, a hybrid scheme (see Figure 2) can be implemented, which uses the two tables together to process user queries. The item-level table only tracks the item insertion/removal events, whereas the epc_bitmap-based table keeps consolidated information at periodic checkpoints. The starting checkpoint epc_bitmap entry is merged with the corresponding duration's item level data to answer queries for a specified time period.

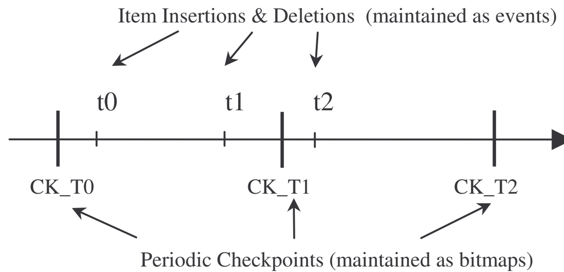

Figure 2: Hybrid Scheme

For historical data, either the same configuration can be retained, or one can maintain only the epc_bitmap-based summary information of appropriate granularity.

However, the epc_bitmap might not be a good candidate for some applications (such as automatic baggage handling, postal mail dispatch), because unlike the retail sector, the items in these applications do not lend themselves well to grouping based on a common property, thus precluding the use of an epc_bitmap for these cases.

3.5 Epc_bitmap-based Auxiliary Structures

Epc_bitmap can also be used to build auxiliary structures on tables with epc_bitmap columns. The basic idea is to create a *summary bitmap* by performing an epc_bitmap operation (OR, AND) on a collection of epc_bitmaps. Subsequently, the summary bitmap can be used to derive information about the collection of epc_bitmaps. For example, for a summary bitmap *sbmap* created using OR operation on $\{bmap_1, \ldots, bmap_k\}$:

$$\text{bmapExists(sbmap)} \rightarrow \exists\ i \in 1 \ldots k\text{: bmapExists}(bmap_i)$$
$$\neg(\text{bmapExists(sbmap)} \rightarrow \forall\ i \in 1 \ldots k\text{: } \neg\ \text{bmapExists}(bmap_i)$$

Thus, indexing structures can be built which exploit these properties of summary bitmaps. Specifically, a *Summary Bitmap Table* consisting of `<scalar column interval, summary bitmap, rid list>`, can be created, where each row holds an OR-based summary bitmap for the scalar column interval and the corresponding set of row identifiers. A query involving `bmapExists()` predicate ANDed with a predicate on the scalar column can now be processed efficiently. For example, the predicate `bmapExists(item_bmap, <epc1>) AND time > <t1>` can be processed by checking for the existence of <epc1> in summary bitmaps corresponding to time > t1 and only for those that evaluate to true, the corresponding set of bitmaps are searched.

Similarly, a *Summary Bitmap tree* can be created by constructing summary bitmaps recursively in a bottom-up manner, from leaf nodes to parent nodes. Use of summary bitmap trees for processing `bmapExists()` queries is discussed in Section 4.

4. Implementing the Epc_bitmap in Oracle DBMS

This section discusses the implementation of an epc_bitmap in Oracle DBMS that supports bitmap indexes.

4.1 Epc_Bitmap and Associated Operations

Oracle DBMS bitmap index implementation manages compressed bitmaps for a collection of table row identifiers. The set of row identifiers are converted into bitmaps using the equality-encoding scheme. Also, Oracle DBMS allows for efficient compression [4, 5] of these bitmaps and supports performing bitmap operations on the compressed format. The DBMS bitmap index code is modified to take in EPC values instead of table row identifiers to support epc_bitmap functionality. Since the EPC is already a unique identifier, its value is directly mapped to a bit position (or bit[epc_suffix_current-epc_suffix_start]=1).

Furthermore, the existing code for supporting table row identifier based bitmaps is based on the fact that all the rowids belong to the same table or table partition. This concept was extended to support the notion of epc_prefixes which was used to group (or segment) bitmaps with a common prefix together into smaller collections.

Epc_bitmap Format: To support an arbitrarily dispersed set of EPCs, epc_bitmap is defined as a multiset type (or nested table type in Oracle) that can hold collections of epc_bitmap_segments, as described in section 2.2. Within each of the epc_bitmap_segments, the shared information (such as epc_length,…) is stored as type attributes. The epc_suffix_bitmap attribute, of type RAW, stores the compressed bitmap representation of a collection of electronic product codes sharing the same prefix (as shown in Figure 1). In Oracle DBMS, maximum size of RAW is 2000 bytes. Thus, a collection of EPCs is grouped into different epc_bitmap_segments, either because their epc_prefixes are different, or because the previous epc_suffix_bitmap could not hold any more EPCs.

The compression of the bitmap is achieved using the byte-aligned bitmap coding scheme [4]. The bitmap is compressed by storing it as sequence of atomic units or bitmap atoms. A bitmap atom holds one control byte (for describing how an atom is organized), zero or more than zero gap bytes (for specifying the distance from the previous atom), and zero or more than zero map bytes (for specifying how the bits are set in the bytes). The detailed compression scheme is described in [4].

Conversion Routines: The epc2Bmap routine generates the epc_bitmap. It determines the format information, namely the type of EPC (64-bit, or 96-bit, or 256-bit), and the segmentation information (prefix and suffix lengths) from the EPC itself. Based on this information, individual epc_bitmap_segments are constructed by segmenting the whole collection of EPCs into smaller collections of EPCs sharing the same epc_prefix. The common information (epc_length,…) is then extracted and the epc_suffixes are converted into a compressed bitmap - epc_suffix_bitmap, using the modified bitmap index code.

The prototype implementation currently handles a homogeneous collection of EPCs, that is, collections of all 64-bit EPCs, or 96-bit EPCs, etc. However, it can be extended to handle heterogeneous collections of EPCs. This can be done by taking the epc_length and the epc_suffix_length into account during the epc_bitmap construction and in the processing of various bitmap operations.

Both bmap2Epc and bmap2Count routines take an epc_bitmap as input. The bmap2Epc constructs the original collection of EPCs, by first generating a set of epc_suffixes from each bitmap atom of the compressed bitmap - epc_suffix_bitmap in each of epc_bitmap_ segments, and combining each pair of (epc_prefix and epc_suffix) into each EPC, and then repeating the above for the next epc_bitmap_segment until the last epc_bitmap_segment. The bmap2Count routine calculates the count of bits set in each bitmap atom of the epc_suffix_bitmap for each of epc_bitmap_segments and returns the sum of the counts.

Other Routines: Because epc_suffix_bitmap is a compressed bitmap representation, the other epc_bitmap routines corresponding to AND, OR, etc. are implemented to take in or return a compressed bitmap format. These routines are more efficient, compared with the embedded bitwise operators that take in and return an uncompressed bitmap format because compressed bitmaps consume much less memory and disk space.Moreover, all bitwise operations are done at the bitmap atom level, i.e. only the interesting bitmap atoms are used, because each bitmap

atom holds the gap information (or position information) that helps determine the interaction between two bitmap atoms from corresponding epc_bitmaps (which obviously must share the same epc_prefix).

4.2 Transient and Persistent Epc_bitmap

Both transient and persistent epc_bitmap can be created. However, persistent epc_bitmap instances require special handling, as described below.

A persistent epc_bitmap is stored in an epc_bitmap column. The epc_bitmap_segments corresponding to an epc_bitmap column are stored together in a separate primary B+-tree structure (i.e. an index-organized table in Oracle [15]), with the key columns: nested_table_id, epc_length, epc_suffix_length, epc_prefix, epc_suffix_start, and non-key columns: epc_suffix_end, epc_suffix_bitmap (see Figure 3). This allows for

- efficient access to epc_bitmap_segments belonging to a row since nested_table_id (shown as `epcbmp_id_1, …, epcbmp_id_n` in Figure 3) is part of the primary key columns.
- utilization of the prefix key compression option on the primary B+-tree to get compact storage.
- preservation of the table row clustering property, i.e., the epc_bitmap column is stored outside of the table, and hence does not impact the table storage characteristics.
- efficient piece-wise maintenance of epc_bitmap_segments, i.e. only the relevant epc_bitmap_segments (instead of all epc_bitmap_segments) need to be maintained during DML, like LOB piece-wise operations .

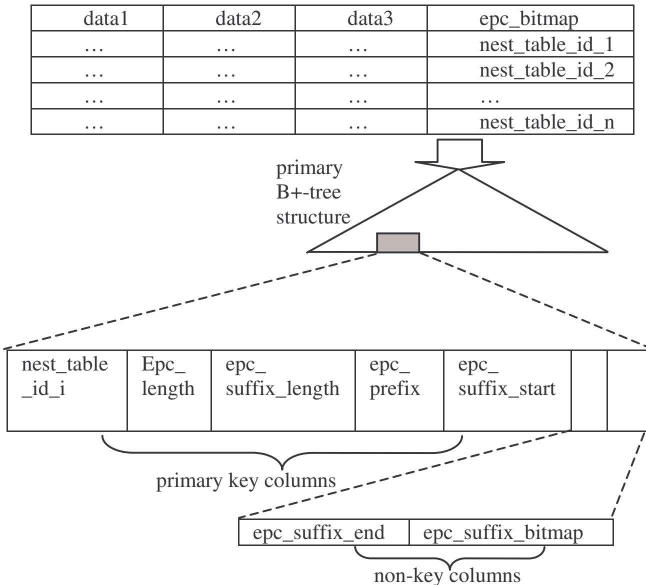

Figure 3: Persistent Epc_bitmap Storage Scheme

4.3 Bulk-load/Incremental Maintenance of a Table with an Epc_bitmap Column

Currently, both bulk-load and incremental maintenance is done at the row level by transforming the input EPC collections into a corresponding set of epc_bitmap_segments (using `epc2bmap` routine) and storing the resulting <nested_table_id, epc_length, epc_suffix_length, epc_prefix, epc_suffix_start, epc_suffix_end, epc_suffix_bitmap> in the associated primary B+-tree. Similarly, the delete operation results in the deletion of the corresponding set of epc_bitmap_segments, and update is implemented as a delete operation followed by insert.

4.4 Query Processing on a Table with an Epc_bitmap Column

Currently, epc_bitmap operations are evaluated in a row-by-row manner by executing the corresponding function on the epc_bitmap instance. If an epc_bitmap operation is used as predicate, for example `bmap2Count(bmap_col) > 5`, a functional B-tree index, which is a B+-tree index built on the value of `bmap2Count(bmap_col)`, can be defined to speed up the processing.

Other special index structures for certain epc_bitmap operations are also conceivable. The following section discusses these ideas. These ideas have not been implemented in the current version and are part of our future work.

4.5 Speeding up Queries Involving Epc_bitmap Column Predicates

As discussed in Section 4.4, epc_bitmap operations are evaluated in a row-by-row manner. However, by creating specialized index structures, one can speed up queries involving predicates on epc_bitmap columns.

Consider this query involving the bmapExists operator:

```
SELECT 1 FROM Shelf_Inventory
WHERE bmapExists(<item_bmap>, <epc1>);
```

To efficiently process such a query, one could build a *summary bitmap tree* structure, which is constructed in a bottom-up manner as follows: each leaf node holds a link (row identifier) to one index column epc_bitmap. The parent level epc_bitmaps are constructed by summarizing over the child node epc_bitmaps. The summarization itself is performed as an epc_bitmap operation (bmapOr in this case). Thus, the root and branch nodes hold links to their children, as well as to the summary epc_bitmap as shown in Figure 4.

To process queries with predicates involving `bmapExists`, the following depth-first search algorithm can be used:

```
DFS(X, epc):
X: a node in the tree index;
epc: a given epc;
bmp(X): the epc_bitmap associated with node
        X;

if ( bmapExists(bmp(X), epc)== TRUE)
{ if (X == leaf node)
    report bmp(X);
  else
    for each child node V of X do
      DFS(V, epc);
}
```

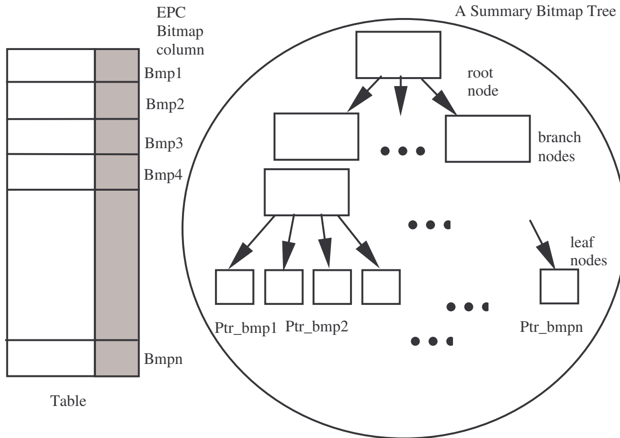

Figure 4: A Summary Bitmap Tree Structure to Speed up the bmapExists Operation

It takes $O(m*\log_m N)$ in the worst case (m: fan-out, N: number of rows) to get the first epc_bitmap candidate, or O(1) to return no candidate. Maintaining the index is also efficient when inserting (or deleting) a row with an epc_bitmap column into (or from) a table. Only parent branch nodes and root nodes need to be updated, using the `bmapOr` operation for inserts and the `bmapMinus` operation for deletes (or `bmapXor` for both operations), all of which are very fast. Occasionally insert operations can cause node split requiring rebuilding of related parent nodes, similar to split operation in a B+-tree index.

5. Performance Evaluation

These performance experiments compare the performance of storing an EPC collection as an epc_bitmap column vs. as a collection column. The analysis and experiments conducted fall into the following categories:

- *Storage Comparison:* These experiments characterize the storage benefits of the epc_bitmap**.**
- *Performance of Bulk-load Operations:* These experiments characterize the performance of loading data for the two configurations.
- *Performance of Queries involving Operations on Collections:* Queries involving COUNT, EXISTS, MINUS, are compared for the two configurations. Performance of logical (AND, OR) operations is similar to that of MINUS operation, and hence is not included in this paper.

5.1 Experimental Setup

The experiments are performed in Oracle 10g Release 1 DBMS, and SunOS 5.8, installed on an Ultra-60 Sparc Workstation with two UltraSparc-II/360 MHz CPUs and 1024 MB of main memory. The database is configured to use 128 MB database cache with a 8KB block size.

Two tables, one for the EPC collection datatype, and the other for epc_bitmap, are defined as follows:

```
CREATE TABLE epc_coll
(time TIMESTAMP PRIMARY KEY,
 epcs epc_table)
```

where `epc_table` is an index-organized nested table type with elements of the form (`epc_value`), which holds individual EPCs. The primary key for the index-organized table is (nested_table_id, epc_value), and

```
CREATE TABLE epc_bmp
(time TIMESTAMP PRIMARY KEY,
 epcs epc_bitmap)
```

where `epc_bitmap` is the index-organized nested table type with elements of `epc_bitmap_segment` type as defined in section 2.2 and described in section 4. These tables represent the shelf analytics scenario as described in section 3.

The data set consists primarily of EPC-64 collections, where the epc_suffix (20 bits) and the lower part of the epc_prefix (4 bits) are randomly generated with a uniform distribution. Because the dataset size is dependent on not only the number of rows, but also the number of EPCs in each row, the two parameters are adjusted to make the final table size comparable. This section presents the results from the following five configurations: A) 720 rows with 100000 EPCs in each row; B) 7200 rows with 10000 EPCs in each row; C) 72000 rows with 1000 EPCs in each row; D) 720000 rows with 100 EPCs in each row; and E) 7200000 rows with 10 EPCs in each row.

In the configurations A, B, and C, the time interval is set to 1 hour and rows are generated for 30, 300, and 3000 days respectively. For example, 720 (=24*30) rows are generated for 30 days. Further, EPC collections of size 100000, 10000 and 1000 are assumed to typically represent the number of items in a department and/or shelf. In the configuration D, the time interval is set to 1/10 hour and the duration is 3000 days, while in the configuration E, it is set to 1/100 hour and the duration is 3000 days. Other configurations are not included in this paper because the performance trend is apparent from the above configurations.

5.2 Storage Comparison

This experiment demonstrates the storage advantages of the epc_bitmap. When there are 100 or more EPCs in a row, the storage size of the epc_bitmap is 6-8 times smaller than that of the EPC collection type (see Figure 5). Even when there are 10 EPCs in a row, the storage size of epc_bitmap is still 2 times smaller, compared with the EPC collection type. As described above, the datasets used are synthetic with a uniform distribution. Thus, the

storage advantages for real world data sets (i.e. more clustered) are expected to be better.

5.3 Performance of Bulk-load Operations

The bulk load performance is important for the epc_bitmap because the epc_bitmap is most useful in OLAP, where a data warehouse is initially loaded and periodically refreshed. Unlike the collection datatype where no transformation is required, the epc_bitmap will incur a *compressed* bitmap construction cost, since the raw data (i.e. a collection of EPCs) needs to be transformed into an epc_bitmap before it can be inserted into a table.

The configuration B is used in this experiment. Figure 6 shows the experiment results. In summary, although the epc_bitmap's construction phase takes time, the overall bulk load time of the epc_bitmap is still 3.7 times faster than that of the EPC collection datatype. The difference is due to the storage advantages of epc_bitmap and the fact that bulk-load is an I/O intensive operation.

The result for other configurations is similar to this configuration, and hence is not included in this paper. Also, the performance of batch maintenance operations is similar to that of bulk-load operations, and hence is not included in this paper.

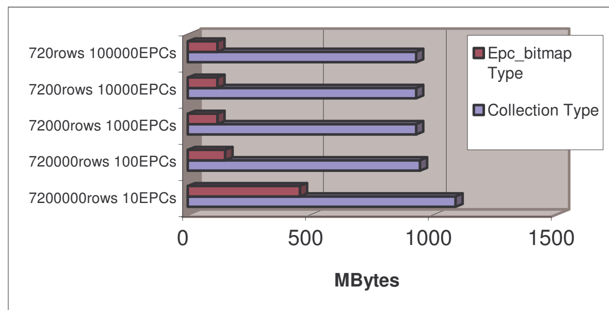

Figure 5: Storage Comparison between Epc_bitmap type and Collection Type

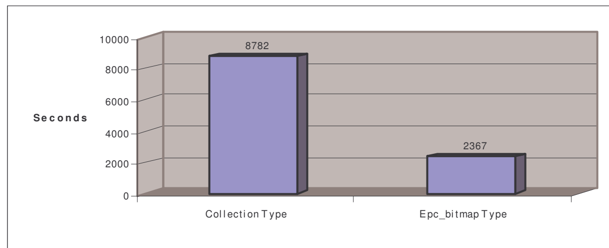

Figure 6: Bulk Load Performance Comparison between Epc_bitmap Type and Collection Type

5.4 Queries involving Epc_bitmap Columns

This set of experiments shows that the query performance of the epc_bitmap is better than that of the EPC collection datatype when there are 1000 or more EPCs in a row. The benefits are due to the compact nature of epc_bitmaps and because most of epc_bitmap operations are significantly faster, especially for EPC collections with 1000 or more EPCs. The results from configurations D and E are not presented because they are similar to configuration C.

5.4.1 Query 1: Count Number of Items Present at a Specific Time

This query is used to count the number of items present at a specific time.

The queries used for the collection type and the epc_bitmap are as follows:

```
SELECT COUNT(*) FROM epc_coll a,
   TABLE(a.epcs) b
WHERE a.time =  '2004-03-04 10:00 AM';

SELECT bmap2Count (epcs) FROM epc_bmp
 WHERE time = '2004-03-04 10:00 AM';
```

Figure 7 shows the experiment results. For the configuration C, there is no difference in query time between the epc_bitmap and the collection type (0.007sec in both cases). But for the configuration B, the query time for epc_bitmap and collection type is 0.017sec, 0.020sec respectively and for the configuration A, the query time is 0.075sec vs 0.175sec respectively.

Thus, when there are 1000 or more EPCs in a row, the query performance using the epc_bitmap is better than the query using the collection type.

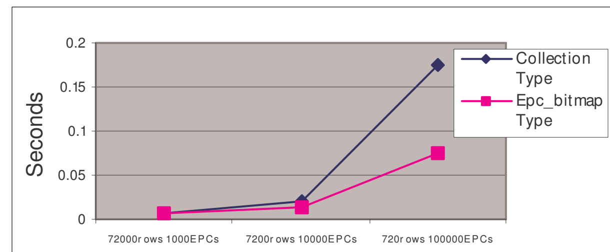

Figure 7: Query 1 Performance

5.4.2 Query 2: Enumerate Items Removed between Two Time Intervals

This query is used to enumerate the items removed from a shelf between two time intervals. The queries used for the collection type and the epc_bitmap are as follows:

```
SELECT b.epc_value FROM epc_coll a,
   TABLE(a.epcs) b
WHERE a.time = '2004-03-04 10:00 AM'
MINUS
SELECT b2.epc_value FROM epc_coll a2,
   TABLE(a2.epcs) b2
WHERE a2.time = '2004-03-04 11:00 AM';

SELECT * FROM TABLE(SELECT
   bmap2Epc(bmapMinus(p1.epcs,p2.epcs))
   FROM epc_bmp p1, epc_bmp p2
   WHERE p1.time = '2004-03-04 10:00 AM' AND
         p2.time = '2004-03-04 11:00 AM');
```

Figure 8 shows the experiment results. In configuration C, there is no difference in query time between epc_bitmap and collection type (0.020sec in both cases). But the query time for epc_bitmap and collection

type is 0.036sec and 0.160sec respectively for configuration B and 0.255sec vs 1.930sec respectively for configuration C.

Thus, for a large collection of EPCs, the epc_bitmap MINUS operation is clearly better than the SQL built-in MINUS operation.

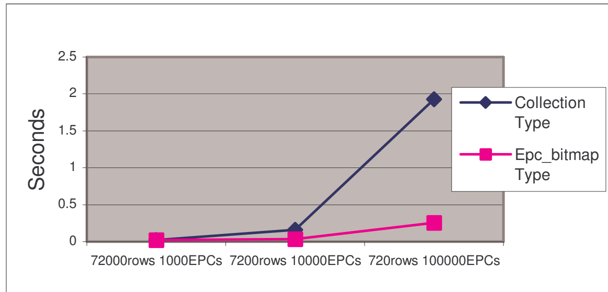

Figure 8: Query 2 Performance

5.4.3 Query 3: Report All Observations When a Given EPC was Present

This query reports all observations when a given EPC was present. The queries used for the collection type and the epc_bitmap are as follows:

```
SELECT a.time FROM epc_coll a
WHERE EXISTS(SELECT 1 FROM TABLE(a.epcs) b
    WHERE b.epc_value = '400003000300052A');

SELECT time FROM epc_bmp
WHERE bmapExists(epcs, '400003000300052A');
```

Figure 9 shows the experiment results. The query times for the epc_bitmap and the collection type are 203sec and 329.5sec respectively for configuration C, 44.3sec and 326.4sec respectively for configuration B, and 25.12sec and 322.5sec respectively for configuration A.

Thus, when there are 1000 or more EPCs in a row, the query using the epc_bitmap is better than the query using the collection type. The query performance of epc_bitmap degrades steeply for configuration C in comparison with configuration B. This performance degradation is due to the execution of epc_bitmap routines in an external process, resulting in context switch between the external process and database process, once for each row processed. If the epc_bitmap operations are natively implemented in the database kernel, the performance in configuration C should be comparable to other configurations.

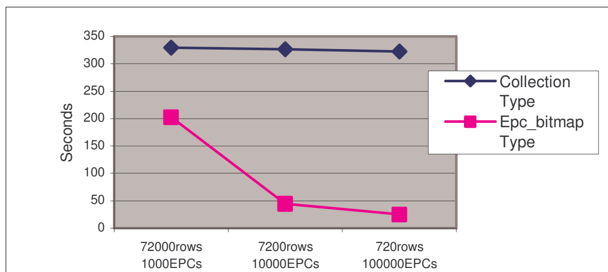

Figure 9: Query 3 Performance

5.4.4 Query 2 with Variant Datasets

Earlier in Section 5.4.2, the epc_bitmap MINUS operation performance was presented for a single dataset. This subsection discusses performance of the epc_bitmap MINUS operation by varying the difference between the two epc_bitmaps being compared. We take configuration B and generate the datasets with {1, 10, 100, 1000, 2500, 5000, 7500, 10000} different EPCs between two time intervals. The query using the collection datatype is as follows:

```
SELECT count(*) FROM
  (SELECT b.epc_value FROM epc_coll a,
     TABLE(a.epcs) b
   WHERE a.time = '2004-03-04 10:00 AM'
   MINUS
   SELECT b2.epc_value FROM epc_coll a2,
     TABLE(a2.epcs) b2
   WHERE a2.time = '2004-03-04 11:00 AM');
```

The query using the epc_bitmap is as follows:

```
SELECT bmap2Count(bmapMinus(p1.epcs,
                            p2.epcs))
FROM epc_bmp p1, epc_bmp p2
WHERE p1.time = '2004-03-04 10:00 AM' AND
      p2.time = '2004-03-04 11:00 AM';
```

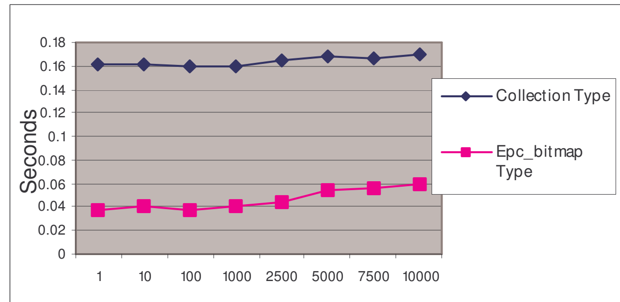

Figure 10: Query 2 with Variant Data Set Performance

The query using the epc_bitmap MINUS/COUNT operation is about 3 times faster than the query using the built-in SQL MINUS/COUNT operation on the collection datatype (Figure 10). The difference is consistently observed for varying datasets. Similar results are observed for other configurations as well (i.e. configurations B and C, etc).

5.5 Discussion

The above experiments were performed with randomly generated data having a uniform distribution. In the real world, where most of the EPCs in a collection are likely to be clustered, the epc_bitmaps are expected to be more compact and efficient, resulting in even better performance.

Also, as noted before, the above scheme was implemented by extending the bitmap indexing technology of the Oracle database management system, and the experimental results are specific to the given implementation. However, the results are generic in nature, and similar results are expected for a bitmap datatype implemented in other database systems.

6. Conclusion and Future Work

The paper makes a case for supporting an epc_bitmap datatype in a DBMS to handle the emerging class of RFID-based item tracking applications. Specifically, RFID tagged items, identified by their electronic product codes (EPCs) can be tracked in groups based on a common property such as their location, container, and expiration date. Such collections of EPCs can be represented by an epc_bitmap, which can be accessed and manipulated using epc_bitmap operations, to build efficient RFID-based item-tracking applications.

The epc_bitmap proposed in the paper can support a variety of classes of EPCs (64-bit, 96-bit, and 256-bit EPCs) and can be used to represent both transient and persistent EPC collections. The epc_bitmap supports conversion, maintenance, comparison and pair-wise logical operations.

A wide range of RFID applications, including retail store shelf analytics, product recall, item returns, and asset management, can be supported using the epc_bitmap, as illustrated in the paper. However, the epc_bitmap might not be a good candidate for the applications that do not lend themselves well to grouping based on a common property.

The epc_bitmap can be easily implemented in a DBMS leveraging the bitmap index code, used to manage bitmaps of table row identifiers. One such implementation was built using Oracle DBMS, and the performance experiments conducted demonstrate both the storage savings and query performance benefits.

Providing epc_bitmap and its associated operations in a DBMS should enable building interesting RFID applications. In the future, we plan to support epc_bitmap's piece-wise maintenance operations and index-based evaluation of queries involving epc_bitmap column predicates. Also, we will perform more experiments (especially on the hybrid scheme described in section 3.4), and explore the similar use of epc_bitmap to support table row identifier collections and life science identifier collections.

7. Acknowledgments

We thank Jay Banerjee and Susan Mavris for their helpful suggestions and for their support.

8. References

[1] Brock, D., Cummins, C., "EPC™ Tag Data Specification", http://www.autoidlabs.com/whitepapers/MIT-AUTOID-WH025.pdf, June 2003.

[2] Engels, D., "The use of the Electronic Product Code™", http://www.autoidlabs.com/whitepapers/mit-autoid-tr009.pdf, February 2003.

[3] Engels, D., "EPC-256: The 256-bit Electronic Product Code™ Representation", http://www.autoidlabs.com/whitepapers/mit-autoid-tr010.pdf, February 2003.

[4] Antoshenkov, G., "Byte Aligned Data Compression", U.S. Patent No: 142640, October 1993.

[5] Jakobsson, H., "Bitmap Indexing in Oracle Data Warehousing", Database seminar at Stanford University. http://www-db.stanford.edu/dbseminar/Archive/FallY97/slides/oracle/, October 1997.

[6] Chan, C. Y., Ioannidis, Y.E., "An Efficient Bitmap Encoding Scheme for Selection Queries", *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pp.215-226, 1999.

[7] Chan, C. Y., Ioannidis, Y.E., "Bitmap Index Design and Evaluation", *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pp.355-366, 1998.

[8] O'Neil, P. and Quass, D., "Improved Query Performance with Variant Indexes", *Proceedings of the ACM SIGMOD Int. Conf. on Management of Data*, pp. 38-49, 1997.

[9] Collins, J., "Wipro Starts Up RFID Retail Pilot", http://www.rfidjournal.com/article/view/1064, July 2004.

[10] Adams, C., "RFID and Supply Chain Theft", http://www.rfidjournal.com/article/view/485, 2004.

[11] Costlow, T., "Product Recalls Spur Move Toward RFID", Design News, http://www.designnews.com/article/CA442376.html?industryid=22213, August 2004.

[12] Wu, K., Otoo, E., Shoshani, A., "On the Performance of Bitmap Indices for High Cardinality Attributes", *Proceedings of the 30th Int. Conf. on Very Large Data Bases*, pp. 24-35, September 2004

[13] Hillyer, M., "The MySQL SET Datatype", http://dev.mysql.com/tech-resources/articles/mysql-set-datatype.html, May 2004

[14] Chawathe, S. S., Krishnamurthy V., Ramachandran S., Sarma S., "Managing RFID Data", *Proceedings of the 30th Int. Conf. on Very Large Data Bases*, pp. 1189-1195, September 2004

[15] Srinivasan, J., Das, S., Freiwald, C., Chong, E.I., Jagannath, M., Yalamanchi, A., Krishnan, R., Tran, A., DeFazio, S., Banerjee, J., "Oracle8*i* Index-Organized Table and its Applications to New Domains", *Proceedings of the 26th Int. Conf. on Very Large Data Bases*, pp. 285-296, Sept. 2000.