EXTENDING SQL TO SUPPORT IMAGE CONTENT-BASED RETRIEVAL

Myrian R.B. Araujo, Agma Traina, Caetano Traina Jr.

Computer Science Department - University of São Paulo at São Carlos - Brazil
[mrenata, agma, caetano]@icmc.usp.br

Abstract

This paper shows a technique to support images in a relational database, so it can fulfill the requirements to be used as the storage mechanism of a PACS (Picture Archiving and Communication System). This support includes the ability to answer similarity queries based on the image content, allowing fast image retrieval based on indexing structures. The main concept enabling this support is the definition of distance functions based on features, which are extracted from the images as they are stored in the database. The proposed extension to the SQL language enables the construction of an interpreter that intercepts the extended commands and translate them to standard SQL, allowing the technique to be used with any relational database server. This extension incorporates the image type to be used as a native data type inside relational database, and provides resources to enable the retrieval of images based on their contents.

Key Words - Content-based image retrieval, Extended SQL, feature extraction, image databases

1. Introduction and motivation

Relational databases (RDB) store large amounts of information in tables composed of attributes of various types, and using RDBs lead to flexible, expandable and highly maintainable information systems. Unfortunately, relational databases do not support images adequately. Images are stored as “Binary Large OBjects - BLOBs” data, allowing their retrieval through textual or numeric keys. However, it is not enough for a great number of systems dealing with images, e.g., medical applications including several kinds of exams, such as X-ray, magnetic resonance imaging (MRI), etc. For medical analysis, it is more important to retrieve images which are similar to a given one. This happens whenever the physician needs to compare some previous case or even to check diagnosis. Therefore, images should be efficiently stored and retrieved allowing content-based techniques. To satisfy the needs of the content-based image retrieval (CBIR) systems, an extension of the standard SQL was carried out [2]. It was implemented to be naturally integrated in applications which execute operations that already exist and access traditional data such as the administrative software or digital patient record systems [3]. Thus, standard SQL should be extended to build and widen applications that already access traditional databases to allow their support to images. For this reason, it is impossible to adopt systems such as QBIC[8], Virage[9], and others, as they do not allow interacting with other applications that are already being operated.

Medical application software is constructed to process images originating from specific equipment or kind of exams, so requests looking for image sets to be processed by a specific application are a very common operation. Reading large sets of full featured image files are costly operations, so the PACS [5] extract values from specific tags of each image as they are being stored in the system, into directory tables. When a query requests images based on tag values, the directories are used to select the required images, speeding up the query answering process. However, the choice of what tags to maintain in the directories is cumbersome, making the whole system inflexible, difficult to improve or expand, and hard to maintain. Moreover, operations of selecting images using content-based retrieval techniques are orthogonal to operations of selecting images using the tag-based approach, leading to duplication of efforts, thus further reducing the efficiency and the flexibility of the system [2].

This paper shows how to improve a relational database to support images through an SQL extension, so it can be used as the underlying storage mechanism of a PACS. It also shows that the resulting system presents not only the desired properties of flexibility, expansibility and high maintainability provided by relational systems, but also maintain the consistency of image distribution presented by PACS, and supports efficient content-based image retrieval operations.

The rest of this paper is organized as follows. Section 2 gives an overview of the SQL standard evolution. Section 3 describes the concepts required to support images in a relational database, and section 4 describes the proposed extensions to SQL to support images. Section 5 presents an architecture to create a system supporting images as a native data type. Finally section 6 presents conclusions of the paper.

2. Background

A great effort to reach standardization has been done lately in several fields. However, new and constant technological and scientific advances in current applications strongly

---

1This research has been partially supported by Fapesp (Fundação de Amparo à Pesquisa do Estado de São Paulo, Brazil - grant 98/09880-1), and by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico, Brazil - grants 52.1685/98-6, 860.068/00-7 and 35.0852/94-4).
demand that robust systems should be developed to support these advances.

Nowadays, many extensions of the SQL standard are being proposed. These extensions aim to incorporate new facilities that are not implemented in official standardization, such as ISO and ANSI. Some of the main examples of these extensions are: SQL/SDA, a query language to support Spatial Data Analysis which can provide easy and effective access to spatial analysis functionality to support decision-making processes based on geo-referenced data [10]; DB2 Spatial Extender, a new technology that brings in a number of key capabilities such as standard-based geometry model and geometry functionality, spatial indexing, etc.[1]; OLE DB DM, a new Application Programming Interface (API) for data mining, proposed by Microsoft, as an extension of the OLE DB standard [13]; WebSSQL, a Query Language for Multimedia Web Documents to retrieve Web pages, where users can specify their search conditions much more precisely, resulting in, more useful information and much less irrelevant results [18]; and finally SQL/LPP, a language extension, which permits descriptive pattern-based queries of time series segments (time intervals) instead of individual records [14].

On the other hand, over the last two decades the main committees of standardization have made great effort to reach the SQL standard. During this period the SQL-86, SQL-89, SQL-92 and SQL-99 standards have been published [7]. To support the great variety of current applications, some additional parts of the SQL standard have already been developed (such as SQL/Framework, SQL/OLB - Object Language Bindings) while others are in development (such as SQL/ MED and SQL/MM) to be included in the SQL-99 standard. The main extensions being researched are shown as follows.

SQL/MED - SQL (Management of External Data) has been projected in such a way that data existing in ordinary files as magnetic tapes in non-relational databases could be used in a relational database [12].

SQL/MM - SQL (Multimedia and Application Package) is another standard based on SQL with structured user-defined types. This standard is divided into multiple parts that have been independently developed: Full-Text, Data Mining, Spatial and Still Image [11]. SQL/MM Full-Text supports storing textual data. This standard allows complex search patterns, including searching for text which includes specific words, words with similar definitions, and so forth.

SQL/MM (Data Mining) standardizes four different techniques: The rules model, the clustering model, the regression model and the classification model. SQL/MM Spatial includes structured user-defined types and associated methods of applications needing to store, manage and retrieve information based on aspects of spatial data. Finally, the SQL/MM Still Image part provides structured user-defined types that allow storing new images in a database, retrieving and modifying them in various ways, and locating them by applying visual predicates to its collection of images. The predicates are represented through a structured type called SI_StillImage, where some methods can be applied to SI_StillImage instances, including scale, rotation, and creation of thumbnail images. Another group of data types allows describing various features of image-types, such as SI_AverageColor, SI_ColorHistogram, SI_PositionalColor and SI_Texture.

However, as the extensions above mentioned do not give support to answer similarity queries based on the content of the data stored, we developed a new extension, which is presented in section 4. When designing the extensions, special attention was paid to minimize the modifications in the language, at the same time maximizing its power to express image retrieval conditions.

3. Concepts for image content-based retrieval

Retrieval of images based on their contents is a high-time consuming process, so usually comparisons are made through a set of characteristics that describes the image and not the image itself. Although these characteristics can be strings given by human specialists, the use of automatic algorithms provides better results, removing subjectivity from the process, and allows processing of a much larger amount of data.

Definition 1: An extractor is an image processing algorithm that, applied over one image, extracts a set of numeric and categorical values, used to describe the image in place of the image itself in further comparison operations between images.

Definition 2: An image feature vector (or feature vector) is a subset of the values extracted by one or more extractors from one image.

Feature vectors can be used to compare pairs of images, enabling a preliminary filtering pass over the set of images. Whenever a new image is stored in the database, a number of extractors obtain its feature vectors, which are stored as numeric or categorical attributes together with the image itself. When images needs to be retrieved, the search mechanism tries to find the images that are most similar to those requested in the query, using the feature vectors of the stored images and of the images provided in the query.

To support image retrieval based on similarity of image content, it is necessary to define what similarity means. This is made through a Distance Function.

Definition 3: A Metric Distance Function over two images - $DF()$ - is an algorithm that compares the feature vectors of two images satisfying well-defined algebraic properties of metric domains (the non-negativity, symmetry and the triangular inequality$^2$), and returns a nonnegative value that is smaller as more similar the two images are.

$^2$ Given three objects $a$, $b$ and $c$ in the object's domain, the triangular inequality property holds for the distance function $d(x, y)$ if $d(a, b) < d(a, c) + d(b, c)$. 
As $DF(i)$ is constructed using feature vectors, there can be one or more $DF(i)$ associated to a specific set of images, such as a $DF(i)$ based on their histograms [4], number of object occurrences and object placement [6], etc. Each $DF(i)$ uses one feature vector.

There are two main similarity comparison operators used to retrieve images: $k$-nearest neighbors and range operators. Given a reference image $Ref_k$, the $k$-nearest neighbor operator returns the $k$ images most similar to $Ref_k$, and the range operator returns the images similar to $Ref_k$ up to a given similarity degree (range).

4. The extended SQL

Maintaining the images in metric domains enables using metric access methods. This greatly improves the speed up of image retrieval. Current database managers use indexing structures like B-trees and R-trees to speed up queries. However, data in metric domains only provides the objects itself and the distances between them, but no ordering or spatial properties so the previous structures do not apply. In such cases metric access methods are the only suitable [16].

To support image content-based queries, the SQL extensions presented here enable the definition of the following concepts: the definition of images as a native data type; the definition of the feature vectors and the corresponding $DF(i)$; the similarity conditions based on the $k$-nearest neighbor and range operators; and the creation of index structures based on the feature vectors and $DF(i)$.

In our approach, images are another data type supported natively by the DBMS, so the first extension in SQL is to allow the definition of STILLIMAGE as a native data type in the CREATE and ALTER TABLE commands. Therefore, images are stored as attributes in any relation that requires them, each relation having any number of STILLIMAGE attributes. Using a simple example, consider a patient record stored in a relation named Patient having a Name and two image attributes, one for his/her frontal mug shot called FrontView, and a LungXray image. Images are declared using the new data type STILLIMAGE in the table creation command as:

```
CREATE TABLE Patient ( 
    Name CHAR(30), 
    FrontView STILLIMAGE, 
    LungXray STILLIMAGE); 
```

Further query commands involving images can be detected through the attributes previously defined as being the STILLIMAGE data type.

4.1 The metric command

The comparison between two images requires the definition of a metric distance function. This is done through the new CREATE METRIC command in the extended SQL. It is included in the DDL part of SQL, and enables the specification of the $DF(i)$ by the domain specialist. Each $DF(i)$ is associated with at least one image attribute, and an image attribute can have any number of $DF(i)$. If at least one $DF(i)$ is associated to an attribute, it can be used in both content-based search conditions and in metric indexing methods. Image attributes not associated to a $DF(i)$ cannot be used in search conditions, so they can be stored and retrieved, but not compared.

A $DF(i)$ definition enrolls all the extractors employed to obtain the corresponding feature vector. If the extractors used in a $DF(i)$ always return the same quantity of features, the feature vectors of every image stored in the associated attribute have the same quantity $N$ of elements. Thus, each feature vector is a point in a spatial domain of dimension $N$. However, extractors may return different quantities of features for different images, generating feature vectors with different quantities of elements, resulting in a non-spatial domain. However this domain can be metric if the $DF(i)$ is metric. To assure this property, only the definition of $L_p$-norm $DF(i)^3$ is allowed over the elements of feature vectors. Those elements correspond to vectors having the difference calculated as the double integral of the curve defined by the vectors [15].

The syntax of the command to define a $DF(i)$ is:

```
CREATE [DEFAULT] [SIMILARITY] METRIC <metric_name> 
ON <Table_name> <ImageAttribute> [USING LP0|LP1|LP2] 
[<ExtractorName><PropertyName><PropertyAlias> 
[<val_weight>]..., <PropertyName> 
<PropertyAlias>[<val_weight>]..., ] 
[<ExtractorName><PropertyName><PropertyAlias> 
[<val_weight>]..., <PropertyName> 
<PropertyAlias>[<val_weight>]..., ...] 
```

The word SIMILARITY is only a running word. It is possible to define one, many or no $DF(i)$ for each ImageAttribute in each Table_name, so they must be identified to be referenced in further commands. The first one defined is used by default in search conditions whenever a metric is not explicitly indicated. The word DEFAULT is used to set a metric as the default.

The clause <metric_name> is used to indicate the name of a metric. ImageAttribute is an attribute whose corresponding data type is STILLIMAGE. The clause Table_name is used to indicate the name of a table that stores attributes in a database.

The optional clause [USING LP0|LP1|LP2] specifies how the distance functions are calculated, i.e., they set the $p$ parameter in the $L_p$-norm as zero, one or two. These functions correspond to the Chebychev, Manhattan and Euclidean metrics, applied over the space of the extracted features [17] - the feature vector. The feature vector that composes a metric is indicated through the extractors used to retrieve them, and is expressed in the clause following the

---

3 A $L_p$-norm distance function of two arrays $X(x_1, x_2, ... x_N)$ and $Y(y_1, y_2, ... y_N)$ is expressed as $DF(X, Y)=\sqrt[p]{\sum_{i=1}^{N} w_i |x_i - y_i|^p}$, where $w_i$ is the weight of each attribute.
Each extractor, i.e. `<ExtractorName>`, can retrieve one or more features, and it is possible to enable some, or all features extracted by each extractor, so the ones used must be listed together with the corresponding extractor, as the `<PropertyAlias>`. The features extracted by each extractor used in the distance function are indicated by the `<PropertyAlias>`, which must be unique to each feature used in every metric defined over this image attribute. A weight can be associated for each `<PropertyAlias>` through the optional clause `<val_weight>`.

The following example illustrates the `CREATE METRIC` command:

```
CREATE METRIC LungHisto ON Patient LungXRay
    (Histogram HistogramAlias INTEGER[256],
    Mini MiniAlias INTEGER,
    Maxi MaxiAlias INTEGER);
```

The `LungHisto` metric is defined for the `LungXRay` attribute (an attribute of type `STILLIMAGE` presented in the `Patient` table). This metric is created according to the `Histogram` extractor and the feature vector obtained is composed of the `Histogram`, `Mini` and `Maxi` parameters, following the definition of the features extracted by the `Histogram` extractor.

### 4.2 The `WHERE` clause

Indexes can be created over an attribute associated to a `DF()`. Whenever a new image is stored in the database, its corresponding feature vectors are also stored in the indexing structure. Therefore, an index is created for each feature vector and `DF()` defined for an attribute. This construct enables the retrieval of images based on more than one similarity criterion, allowing the user to choose which criterion is intended to answer each query. For each image, each indexing structure stores the image identifier (`ImId`) and the corresponding feature vector.

When queries are placed, the feature vectors are compared instead of the images. If the query refers to images not yet stored in the database, the proper feature vectors are extracted and used to search the database. Executions of k-nearest neighbors and range search procedures using an indexing structure return the set of `ImIds` that answers the query. The real images are then fetched, and returned as the answer.

Search conditions are stated in the `WHERE` clause of `SELECT` commands, using both k-nearest neighbors and range query conditions. Thus, a new syntactic construction was included in the `WHERE` clause:

```
<search_condition> =
    <previous existing search conditions> |
    <imgAttr> {NEAREST | FARTHEST} <val>
    [BY <metric_name>] [RANGE <val_range>]
    [STOP AFTER <n_nearest> [WITH TIE LIST]]
```

The search condition looks for images among the attributes of type image indicated by `<imgAttr>`. The modifiers of this search condition allow retrieving the objects that are nearest to or farthest from the central query object, which is indicated as the `<val>` value. This value is a reference to an image, the central query object of this command. There are three ways to reference an image: indicating an external file ("File<path>") containing an image, as the result of a sub-query command returning one image, and as the result of an image processing user defined function. The `BY` modifier allows the specification of the `DF()` among those defined for this `<imgAttr>` attribute. If a `BY` modifier is not specified, the default `DF()` is used.

The `RANGE` modifier indicates a similarity limit, indicated by `<val_range>`, thus implementing a range query condition. The `STOP AFTER` modifier allows indicating a limit for the number of objects retrieved in this search condition, thus implementing a k-nearest (or k-farthest) neighbor query condition, where `<n_nearest>` is the value of k. The `STOP AFTER` modifier also imposes an ordering of the retrieved objects, using its distance from the query center object. The `[WITH TIE LIST]` option can be used with the `STOP AFTER` modifier, allowing to retrieve more than `<n_nearest>` neighbors if there are ties among the last objects in the similarity-sorted query answer.

At least one of the `RANGE` and `STOP AFTER` modifiers need to be indicated. If both are indicated, the retrieved objects should satisfy both conditions, besides being ordered through their similarity to the central query object. The query center is constant and can be specified in one of three distinct ways: as a result of a sub-query, as an external file containing an image or as the result of an image processing operation over other images.

Example 1. The following command retrieves the images similar to `LungXRay_Ref` up to a given similarity degree (range). In this case it will be retrieved images similar up to 3 units from `LungXRay_Ref`.

```
SELECT * FROM Patient WHERE LungXRay_Ref NEAREST LungXRay.jpg RANGE 3;
```

Example 2. The following command retrieves the 10 images in the `Patient` relation most similar to the `LungXRay` of the patient “John Doe”.

```
SELECT Name, LungXRay FROM Patient WHERE LungXRay NEAREST (SELECT LungXRx FROM Patient WHERE Name= 'John Doe') STOP AFTER 10;
```

### 5. An architecture to support image retrieval

Existing PACS were built to centralize the storage of images, providing a way to retrieve them based on well-defined protocols, and to distribute them to client workstations or processes. These systems are not designed to be integrated with other existing systems, so it is cumbersome to incorporate them into other applications already running in a medical center [3]. On the other hand, those applica-
tions uses relational database systems (RDBMS), what makes easier to integrate different applications using SQL. As RDBMS do not support content-based image retrieval, we developed a layer atop a RDBMS that monitors the communication stream between the applications and the database server. This layer, called the Content-based Image Retrieval Core Engine - CIRCE, embraces the extended version of SQL as presented in the previous section.

The general architecture of CIRCE is shown in figure 1. This figure shows three conceptual databases - the ADB, IPV and IDD - although physically all of them are in the same database. The ADB corresponds to the application database, such as the hospital administrative system or traditional patient record systems. Existing applications do not support images, so the attributes of the relations stored in the ADB are only numbers and texts. The ADB can be queried using either the standard SQL or the extended SQL. However, when a new application uses images, or when an existing one is expanded to support images, it must use extended SQL through CIRCE.

![Figure 1. Architecture of CIRCE](image.png)

Each attribute STILLIMAGE defined in a relation establishes a set of images disjoint from the set established by other STILLIMAGE attributes. If a query command uses only non-image attributes, the command is sent untouched to the ADB. Whenever an STILLIMAGE attribute is mentioned in a query command, CIRCE uses the IPV and IDD databases, together with the metric indexing structures and the feature extractors, to effectively implement image retrieval by content.

The IPV and IDD databases hold information about each image attribute defined in the application database. For each one a new relation is created in the IPV database, containing two attributes: a blob attribute storing the actual image, and an Image Identifier - ImId, created by the system. The identifier is a code number, unique for every image in the database, regardless of the relation or attribute where it is stored. Each CREATE TABLE command referencing STILLIMAGE attributes is modified, and a numeric data type replaces the STILLIMAGE data type in the modified command sent to the ADB. The corresponding IPV relation is named after the concatenation of the table and attribute names of the original image attribute. In this way, occurrences of this attribute in further query commands are intercepted by CIRCE and translated accordingly. For example, the Patient relation with two STILLIMAGE attributes generates two relations in the IPV database named Patient_FrontView and Patient_LungXRay, each one with two attributes: the image itself and the ImId.

When a DF() is defined for an STILLIMAGE, the corresponding relation in the IPV relation is modified to add the elements of the feature vector as numeric attributes. The IDD database is the schema for STILLIMAGE attributes, and stores information about the extractors and the feature vectors. This database also guides the system to store and retrieve the attributes in the IPV database. Whenever a tuple containing STILLIMAGE attributes is stored, each image is processed by the set of extractors in the XP module, following the definitions retrieved from the IDD database. After that, its ImId is created and stored, together with the image and the feature vector, in the IPV database. The original tuple is stored in the ADB, switching the images with the corresponding ImId identifiers. Using these ImId, the IPV database is used to retrieve the actual images that, in turn, are passed as the answer to the requester process.

![Figure 2 - Screen shot of the IISQL tool](image.png)

To support the development of applications that use the image-enabled extension of SQL, we developed an image-enabled version of the well known Interactive-SQL (ISQL) tool, called Image-enabled Interactive-SQL (IISQL). Aiming to provide a seamless transition to databases that support images, the IISQL was implemented resembling the graphic version of the ISQL, commonly used in the windowed environments. Like ISQL, the IISQL tool presents a window where the user can type commands in extended SQL, and another window where the numerical and textual attributes of the results of the queries are shown. However, IISQL augments this interface with a third window, where the image attributes of tuple selected in the second window are shown. Figure 2 shows an screen capture of IISQL showing a simple query over the Patient relation. The images stored in this relation are real MR tomographies from the hospital participating in this research, although the other data are artificial, to assure the data are anonymous, as is required for this kind of
information. The indexing structure used is the Slim-tree [16], one of the most efficient metric access methods nowadays.

6. Conclusions

In this paper we described a new and powerful extension of the SQL to support content-based image retrieval. We also presented an interpreter developed to answer queries, based on this extension.

As far as the authors are concerned, this the first system to integrate content-based image retrieval with a relational database in an open architecture that can be integrated to existing operational or administrative databases. Moreover, it can accept the definition of new comparison methods and image analyzers (extractors) at any time, in a seamless manner.

This extension was designed aiming to minimize modifications in existing applications, when they need to be extended to take advantage of the new image support. Special attention was also paid to both minimizing the impact of the alterations in the SQL standard, and maximizing the power and efficiency of querying images by similarity. In fact, only one new command was included into standard SQL - the \texttt{CREATE METRIC} command, together with new clauses in three other commands. Besides the small syntactic modifications, similarity queries based on $k$-nearest neighbors and range can be fully expressed, together with a flexible and powerful capacity to define similarity measures.

Finally, the extended SQL interpreter, known as CIRCE, supports the storage of images as a native data type and their indexing and retrieval based on their content through similarity search operators. Written in Borland C++ Builder, it was tested with Oracle and Interbase database managers.

7. References