SIREN: A Similarity Retrieval Engine for Complex Data

Maria Camila N. Barioni Humberto Razente Agma Traina Caetano Traina Jr.

Computer Sciences Department – ICMC/USP
Caixa Postal 668, 13560-970, São Carlos, SP, Brazil
{mcamila, hlr, agma, caetano}@icmc.usp.br

ABSTRACT

This paper presents a similarity retrieval engine – SIREN – that allows posing similarity queries in a relational DBMS using an extended syntax that adds the support for such type of queries in the SQL language. It discusses the main architecture of SIREN, describes some key features and provides a description of the demo.

1. INTRODUCTION

As objects of complex types such as multimedia data (e.g. images, audio tracks, video and long texts), geo-referenced information, time series, fingerprints, genomic data and protein sequences, among others, are being stored at an increasing rate in Relational Database Management Systems (RDBMS), the need to support similarity queries also increases [3]. However, the standard SQL query language does not provide effective support for such queries. Currently, the International Standards Organization is working in the SQL/MM [7], a multi-part standard proposal that provides storage and manipulation support for multimedia data based on user defined types and functions (UDT and UDF). Commercial products such as Oracle InterMedia and IBM DB2 IAV Extenders follow this approach. While this extension can use the existing highly optimized algorithms for each specific similarity operation, it does not allow optimizations among the operators nor their integration with the other operators used in a query.

Supporting to similarity queries from inside SQL in a native form is important to allow optimizing the full set of search operations involved in each query posed. Therefore, integrating similarity queries in a fully relational approach, as proposed in this paper, is a fundamental step to allow the supporting of complex objects as "first class citizens" in modern database management systems. This paper presents an extension to the SQL and a similarity retrieval engine that allows posing similarity queries in a relational DBMS. Table 1 compares our proposed approach to representative available systems that allow posing similarity queries.

VLDB '06, September 12-15, 2006, Seoul, Korea.

Table 1: Comparison of similarity query approaches.

Features	SIREN	Oracle*	DB2**
Representation of similarity queries	Native predicates	Ranking functions	Ranking functions
Optimizations among similarity operators and/or relational operators	Yes	No	No
Multiple distance functions when defining similarity measures	Yes	No	No
Inclusion of new extractors	Yes	Yes	Yes

* Oracle Intermedia 10g R2. ** DB2 AIV Extenders V. 8.

SIREN (*SImilarity Retrieval ENgine*) was implemented to evaluate the adequacy of a syntax extension [1] and it can be accessed at [5]. SIREN acts like a blade between a conventional DBMS and the application programs intercepting every SQL command sent from the application. If it has no similarity construction nor a reference to complex objects, SIREN sends the command to the underlying DBMS and sends back the answer from the DBMS to the application program. So, when only conventional commands are posed by the application, SIREN is transparent. When the SQL command has similarity-related demands or references to complex objects, the command is re-written to execute the similarity-related operations internally, using the underlying DBMS to execute the conventional data operations.

Our demo illustrates the key features and applications of SIREN. In particular, we present: how similarity queries can be supported in SQL; how complex objects are manipulated, stored, indexed and retrieved; the description of a web-based prototype connected to SIREN that provides an environment that allows posing SQL commands (both standard and extended); and an illustration of some SQL commands that can be employed to explore complex data by similarity.

2. SIREN DESCRIPTION

SIREN is composed of three main components (see Figure 1): the interpreter of the language extension that adds similarity queries to SQL; the feature extraction algorithms employed to extract features to represent and index complex objects; and the access methods developed to answer similarity queries (Metric Access Methods – MAM). The main aspects related to each one of these components are briefly described in the next sections.

2.1 Supporting Similarity Queries in SQL

To allow the representation of similarity queries over complex domains, it is necessary to define the data types where

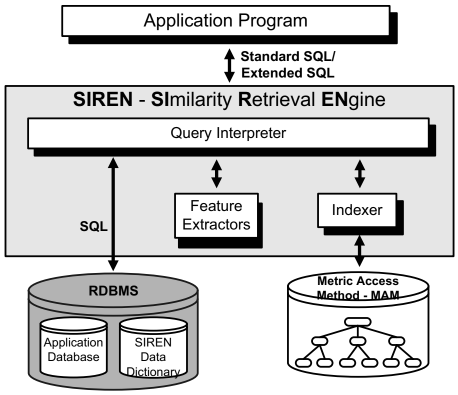

Figure 1: SIREN architecture.

similarity will be measured. Two kinds of complex domains were considered: those stored as structures composed of any number of traditional attributes in a relation (e.g. time series, geo-referenced information, etc.) and those monolithically stored as Binary Large Objects – BLOBs (e.g. images, audio tracks, etc.). They are represented respectively by the domains `PARTICULATE` and Monolithic. In this paper, we illustrate Monolithic domains using images. Other examples of Monolithic domains are audio and video. Thus, new data types were defined, such as `PARTICULATE` and `STILLIMAGE`, where the last one is a specialization of the domain Monolithic regarding the manipulation of images.

A core issue on including similarity queries in SQL is how to define similarity measures (metrics). As there is no concept resembling the definition of comparison operators in SQL, it is needed to create new commands. Similarity measures are stored in the database catalog, thus their manipulation commands should follow the DDL command style. Hence, we created three commands to handle distance functions: the `CREATE METRIC`, the `ALTER METRIC` and the `DROP METRIC`. A metric embodies every handling of the attributes employed to calculate similarity, including feature extraction (for monolithic domains), attribute normalization and the object comparisons.

Once the metrics have been created, they can be associated with one or several complex objects defined as attributes in any relation as a constraint for the attribute, following the two usual ways to define constraints in a `CREATE TABLE` command: as a column constraint or as a table constraint. Moreover, as they enable the creation of indexes to speedup queries, they can also be defined in a `CREATE INDEX` command.

The syntax of the DML commands – `SELECT`, `UPDATE` and `DELETE` – were also extended with new constructions that allows expressing similarity predicates. The syntax of the `INSERT` command does not need changes, although its implementation must consider the new data types. Some examples of the use of the new syntax are presented in Section 3. The complete syntax of the extension to SQL can be found in [5]. The proposed extension aimed at causing a low impact on SQL, whereas providing a strong support to represent similarity queries.

2.2 Handling complex objects in SIREN

`PARTICULATE` data types are stored by their constituent parts, presenting no new storage requirements. On the other hand, the storage of `STILLIMAGE` and `AUDIO` data types must include the features associated with them. Although these data types are stored in attributes of type BLOB, it is also required to store their extracted features. Feature extraction is usually costly, and it must be executed for each object once, when the object is stored in the database. The features are stored as textual or numeric attributes associated with the complex object. As the user does not provide attributes in the relations to store the extracted features, the system must provide the places to store them and their association with the BLOB data in a way transparent to the user. This is similar to what most commercial DBMSs do in order to store BLOB data.

To store the features extracted from a `STILLIMAGE` object, SIREN changes the definition of user defined tables that have `STILLIMAGE` attributes as follows. Each `STILLIMAGE` attribute is changed to a reference to a system-controlled table that has as its attributes the `BLOB` and a set of attributes that stores all features got by every extractor used in each metric associated with the attribute. A new table is created for each `STILLIMAGE` attribute. Whenever a new image is stored in the database, SIREN intercepts the `INSERT` command, stores the non-complex attributes in the user table and the images in the corresponding system tables. Then, SIREN calls the feature extractors and stores their outputs in the corresponding system tables. The storage of the features as regular attributes in hidden relations, instead of BLOB tags as it is done by Oracle Intermedia, allows better indexing and retrieval operations and also a consistent behavior over Monolithic and `PARTICULATE` data types.

Whenever the user asks for data from its tables, SIREN joins the system tables and the user tables, removing the feature attributes, so the user never sees the table split nor the features. Figure 2 shows an illustration of how these new data types are stored by SIREN.

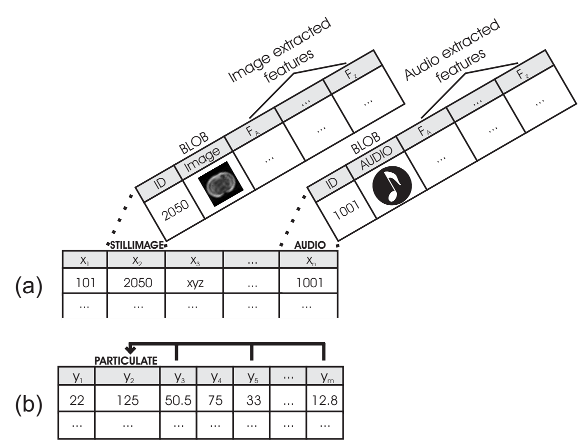

Figure 2: Storage schema of the new complex data types. (a) `STILLIMAGE` and `AUDIO` data types. (b) `PARTICULATE` data type

When the user poses queries involving similarity predicates, SIREN uses the extracted features of the Monolithic objects or the set of attributes that compose `PARTICULATE` objects to execute the similarity operators. The current version of SIREN has three types of feature extractors regarding the STILLIMAGE data type (although other extractors can be easily included): a texture extractor (TEXTUREEXT), a shape extractor based on Zernike Moments (ZERNIKEEXT) and a color extractor based on the normalized color histogram (HISTOGRAMEXT) [6].

SIREN implements five similarity operators. The first two perform the similarity selection and correspond to the two traditional types of similarity search: the *Range query* (Rq) and the *k-Nearest Neigbor query* (k-NNq) [3]. The other three operators implement the similarity joins: *Range Join*, *k-Nearest Neigbors Join* and *k-Closest Neigbors Join* [2].

The metric access method (MAM) employed by SIREN to index PARTICULATE and STILLIMAGE attributes is the Slim-Tree [8], available in the Arboretum (an open source C++ library which implements various MAMs) [4]. The similarity selection operators are already developed in Slim-Tree. However, there is no procedure published to execute the similarity join operators in this or in any other MAM. Thus, if a complex attribute is associated with an index, SIREN executes the similarity selection using a Slim-tree. If the attribute does not have an index, or the required operator is a similarity join, then the query is answered using sequential scan. SIREN has been implemented in C++ and it employs the ODBC protocol to connect with a DBMS. Although the current version of the prototype is running over an Oracle 10g DBMS it can easily be adapted to other DBMS. An eventual implementation of a DBMS with native support for similarity queries can follow the same approach adopted by SIREN.

3. EXAMPLE APPLICATIONS

In order to demonstrate the usability of the similarity retrieval engine, a web (cgi) application was developed in C++ providing an environment that allows posing SQL commands. In this section we use this tool to show examples of posing similarity queries based on the extended language.

In order to illustrate the new data domains defined, two data sets are employed. The first one, called MedImages, is composed of 5,180 medical images. The images are Computerized Tomographies (CT) from three human body parts: abdomen, cranium and thorax. Each tuple of this data set has an image id, the image, the description of the body part and an attribute that specifies whether the image identifies a pathological condition. Similarity queries can be posed to explore this data set considering several aspects of the images such as the similarity based on color distribution or on texture. Two metrics were defined to handle these features, one using the HISTOGRAMEXT extractor and other using the TEXTUREEXT extractor. The commands employed to create the table and the metrics follows.

```
CREATE METRIC HistogramLP1 USING LP1 FOR STILLIMAGE
       (HISTOGRAMEXT (Histogram AS Histo));
CREATE METRIC Texture FOR STILLIMAGE
       (TEXTUREEXT (Texture AS T));
CREATE TABLE MedImages (
       Id INTEGER PRIMARY KEY,
       BodyPart VARCHAR(15),
       Img STILLIMAGE,
       Pathology CHAR(1),
       METRIC (Img) USING
         (HistogramLP1 DEFAULT, Texture));
```

Examples of queries that utilize each one of these metrics are shown in Figure 3.

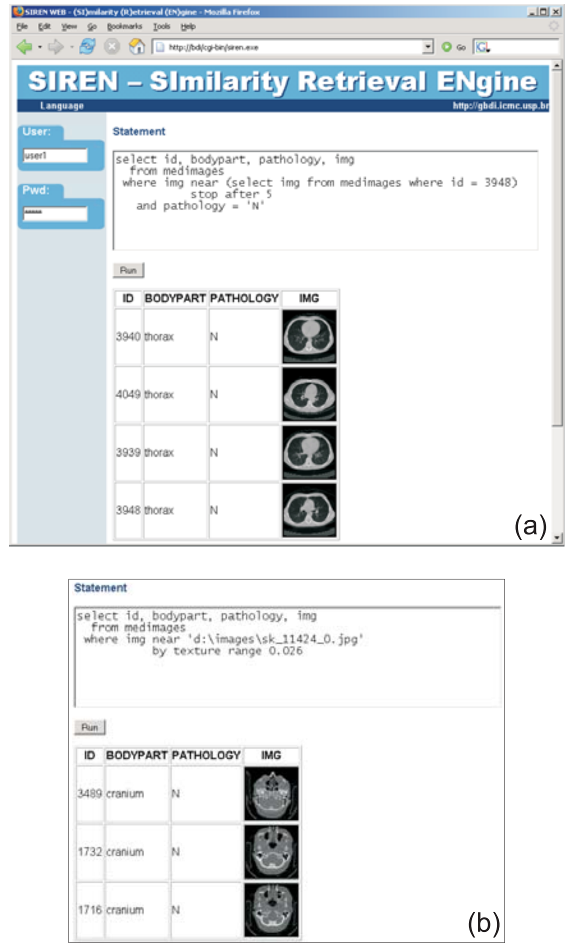

Figure 3: Similarity query examples based on MedImages table. (a) *k-NN query* considering the default metric (Histogram). (b) *Range query* considering the Texture metric.

The second data set, called Cars[1], is composed of the description of 392 cars. This data set is composed of five attributes that describe the following variables: MPG (miles per gallon), horsepower, time to accelerate from 0 to 60 mph (sec.), origin of car (American, European or Japanese) and the car names. This data set can be queried by similarity considering several questions. Here we show a metric to compare cars based on the cost-benefit ratio related to the variables horsepower, acceleration and MPG. The commands employed to create the table and the metric described above are presented following.

```
CREATE METRIC CostBenefit
       USING LP2 FOR PARTICULATE
       (hp FLOAT 5.0, mpg FLOAT, sec FLOAT 10.0);
CREATE TABLE Cars (
       CarName CHAR(35),
       Horsepower FLOAT,
```

[1]This is the cars.data data set available at http://lib.stat.cmu.edu/

```
    Consumption FLOAT,
    Acceleration FLOAT,
    Origin CHAR(8),
    Car PARTICULATE,
    METRIC REFERENCES (Horsepower AS hp,
                       Consumption AS mpg,
                       Acceleration AS sec)
           USING (CostBenefit DEFAULT));
```

Several metrics can be associated with each complex attribute. Using the metric associated with the Cars data set, it is possible to perform queries such as those presented in Figure 4.

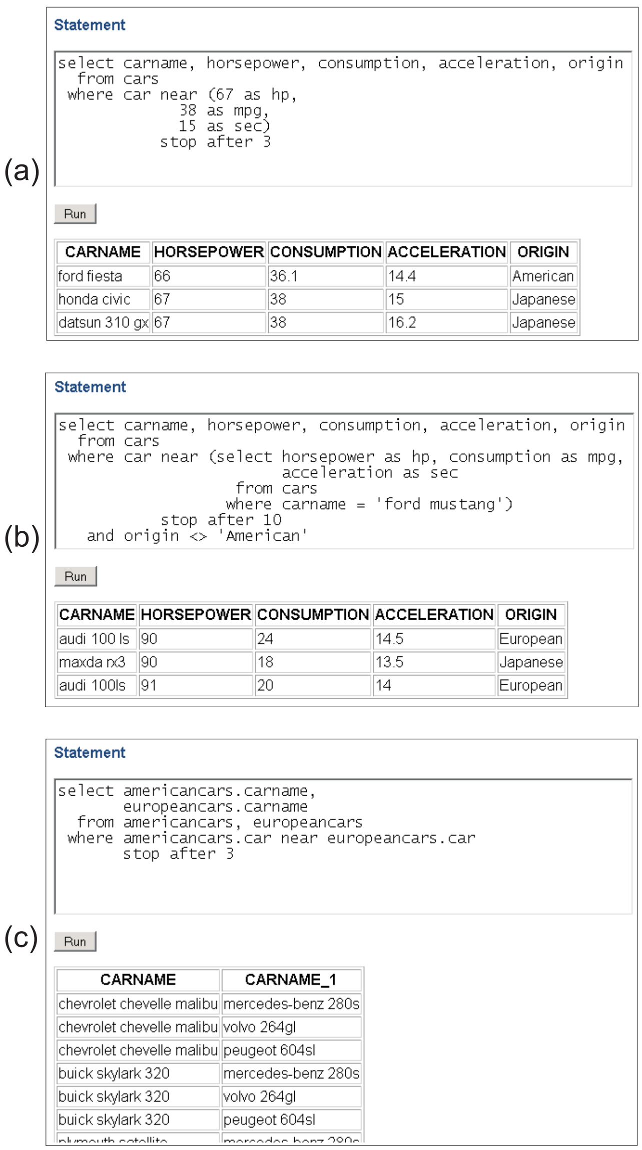

Figure 4: Similarity query examples based on the Cars table. (a) "Which are the 3 most similar cars having: Horsepower = 67 hp, Consumption = 38 mpg and Acceleration = 15 s?". (b) "Which are the 10 cars most similar to the given car and whose origin is not American?". (c) "Which are the 3 most similar European cars to each American car?".

Considering the execution of a query, the time spent by the search in a MAM depends on several factors such as: the size of the feature vector; the number of objects indexed; and the time needed to read a block in a hard disk. As a reference, the total time spent by SIREN to analyze and execute the query presented in Figure 3(b) was 0.29 seconds and in Figure 4(b) was 0.16 seconds, running in a 1.8 GHz PC (including the time spent to read the data dictionary, but not the time spent to generate the image thumbnails and the preparation of the html page).

4. CONCLUSIONS

This paper presented SIREN, a SQL interpreter prototype that allows executing similarity queries over complex data stored in RDBMS. The tool allows executing selections and joins based on the similarity among objects of a complex type. Two kinds of complex attributes are defined: image, and complex attributes defined by simple attributes. The tool allows executing simple queries as well as any combination of operations among themselves and among traditional selections and joins, providing a powerful way of executing similarity queries in complex databases. In addition to the example applications presented herein, there are other data sets and several query examples that can be executed in a web-based prototype connected to SIREN that is available at http://gbdi.icmc.usp.br/siren.

5. ACKNOWLEDGMENTS

This work has been supported by FAPESP (São Paulo State Research Foundation), by CNPq (Brazilian National Council for Supporting Research) and by CAPES (Brazilian Coordination for Improvement of Higher Level Personnel).

6. REFERENCES

[1] M. C. N. Barioni, H. Razente, C. Traina-Jr, and A. J. M. Traina. Querying complex objects by similarity in SQL. In *Brazilian Symposium on Databases (SBBD)*, pages 130–144, Uberlândia, MG, 2005. SBC.

[2] C. Böhm and F. Krebs. High performance data mining using the nearest neighbor join. In *IEEE International Conference on Data Mining (ICDM)*, pages 43–50, Maebashi City, Japan, 2002.

[3] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321, 2001.

[4] GBDI-ICMC-USP. GBDI Arboretum Library. http://gbdi.icmc.usp.br/arboretum/, 2006.

[5] GBDI-ICMC-USP. Similarity Retrieval Engine – SIREN. http://gbdi.icmc.usp.br/siren/, 2006.

[6] F. Long, H. Zhang, and D. D. Feng. Fundamentals of content-based image retrieval. *Multimedia Information Retrieval and Management, Springer*, 2002.

[7] J. Melton and A. Eisenberg. SQL Multimedia and Application Packages (SQL/MM). *SIGMOD Record*, 30:97–102, 2001.

[8] C. Traina-Jr., A. J. M. Traina, C. Faloutsos, and B. Seeger. Fast indexing and visualization of metric datasets using Slim-trees. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 14(2):244–260, 2002.