

2. O: a set of generalization operators, *which are used to construct higher layer relations from lower layer ones*;
3. C: a set of integrity constraints; *and*
4. D: a set of database relations, *which consists of all the relations (primitive or generalized) in the multi-layered database.* □

The first component, a database schema, outlines the overall database structure of an MLDB. It stores general information such as types, ranges, and data statistics about the relations at different layers, their relationships, and their associated attributes. More specifically, it describes which higher-layer relation is generalized from which lower-layer relation(s) and what generalization operator is used. Therefore, it presents a route map for schema browsing and database content browsing and for assistance of intelligent query answering and query optimization.

The second component, a set of generalization operators, provides a set of predefined operators to perform generalizations, which transform lower layer relations to high layer ones.

The third component, a set of integrity constraints, consists of a set of integrity constraints to ensure the consistency of an MLDB.

The fourth component, a set of database relations, stores data relations, in which some of them are primitive (i.e., layer 0) relations, whereas others are higher layer ones, obtained by generalization.

The basic motivation of MLDB is that by generalization relations will become smaller in size. Queries will be answered first by higher layer relations which are smaller and thus faster for mobile devices. The detailed information in lower layers are not transmitted to clients until it is requested.

Our definition of MLDB extends the original definition of MLDB in [2]. The second component of the model is changed from a set of concept hierarchies in the original definition to a set of generalization operators. The only generalization operator in the original MLDB is concept hierarchy climbing. The extension of this component greatly enhances the power of the model. In particular, this allows us to generalize multi-media data, for which concept hierarchy seldom exists.

Several generalization operators are listed as follows. They are by no means an exclusive list. Like the model in [2], we assume the generalization operations are applied to individual attributes, whose value is an object. Different generalization operators may be selected for different attributes and operators may be applied more than once.

- Selection or sampling.

A random or predefined part of an object is used to represent the whole. For example, the first 5 seconds of a video clip or the central quarter of a picture can be used to represent the entire video or picture.

- Resolution.

The object is represented in a coarse scale. For example, an image may be replaced by a thumbnail of the image, which is obtained by grouping local pixels in a template into one.

- Transformation.

The object is transformed into another representation. For example, Wavelet transformation of an image may be used to get an approximation of the image.

- Concept hierarchy climbing.

A primitive concept is replaced by a more general concept using a concept hierarchy, which organized concepts according to their generality levels. For example, an address “1870 Miner Circle, Rolla, Missouri” can be generalized to “Rolla, Missouri”, then to “Missouri”.

- Categorization.

Continuous values are grouped into categories which would reduce the number of values for the attribute. For example, the price can be categorized into \$0-5, \$5-10, \$10-25, \$25-50, \$50-. A particular price can be replaced by its category.

- Aggregation.

Aggregate functions such as min, max, count, sum, and average can be applied to collection objects. The results can be used to replace the collection. For example, the number of videos may be returned to the client instead of the videos.

- Discoloration. By reducing the number of colors used in visual objects (graphs, images, pictures, videos), the objects are generalized.

- Deletion.

The attribute can be simply removed when generalizing from a lower layer relation to a higher layer relation.

Example 1 Suppose a tourist information database contains the following data relations.

```
attraction(name, address, {open_times}, admission_price, description, map, {picture}, {video})
hotel(...)
restaurant(...)
...
```

These relations are at layer 0 in the MLDB. In the following, we use the *attraction* relation as an example to illustrate how an MLDB can be constructed.

At layer 1, the videos of *attraction* are generalized by replacing the videos with several representative still images. The pictures are generalized by replacing the pictures with thumbnail pictures. The map is generalized by using a coarse resolution map. The schema of the generalized relation *attraction*¹ is as follows. *attraction*¹(*id*, *name*, *address*, {*open_times*}, *admission_price*, *description*, *coarse_map*, {*thumbnail_picture*}, {*still_video_image*}).

At layer 2, the videos are further generalized by replacing the still images with the titles of the videos. The pictures are further generalized by replacing the thumbnail pictures with the titles of the pictures. The map is further generalized by replacing the coarse resolution map with directions in text. The schema of the generalized relation *attraction*² is as follows. *attraction*²(*id*, *name*, *address*, {*open_times*}, *admission_price*, *description*, *directions*, {*picture_title*}, {*video_title*}).

At layer 3, the videos, pictures, and map will be removed. The description of *attractions* is generalized by summarizing the description. The schema of the generalized relation *attraction*³ is as follows. *attraction*³(*id*, *name*, *address*, {*open_times*}, *admission_price*, *summary*).

Queries can be answered efficiently and intelligently using the MLDB. For example, a user may ask the information about the attractions in “Boston”. The query can be first answered by using *attraction*³. The user can then select interested places and ask for more information. The further queries will be answered by using *attraction*², *attraction*¹, and finally, *attraction*. This avoids the communication bottleneck by abstracting information into multiple layers and providing detailed information only when it is asked. □

In Example 1, other relations can be generalized as well. Moreover, relations at all levels can participate in join since its primary key or object identifier will not be removed.

3 Building MLDBs

An MLDB is constructed by generalization of the layer 0 (original) database. Since there are many possible ways to generalize a relation, it is practically impossible to store all possible generalized relations. Besides, it is not necessary to materialize all possible generalized relations since some are rarely asked by users. More importantly, too many layers will cause excessive interactions between clients and server which increase the communication cost. Therefore, it is very important to determine the appropriate generalizations to form useful layers of databases.

The basic idea in building MLDBs is to select generalizations based on user interests. The overall cost for an MLDB can be expressed as follows.

$\text{total_cost} = \text{communication_cost} + \text{storage_cost} + \text{maintenance_cost}$
 $\text{communication_cost} = \text{data_transmission_cost} + \text{message_passing_cost}$
 $\text{data_transmission_cost} = \sum_{r \in \text{MLDB}} \text{request_frequency}(r) \times \text{size}(r)$
 $\text{message_passing_cost} = \text{total_number_of_messages} \times \text{size}(\text{message})$

For a large database with many relations and many generalization levels, it is impractical even impossible to find the optimal solution. However, the following heuristics can help us to decide the layers.

- If an attribute is rarely asked by users, it is wise to remove this attribute in a higher layer.
- If one or several levels of generalization are frequently asked by users, they should be materialized.
- If the generalization of several objects are often associated together in queries, they should be put together to form a generalized relation.
- The difference in relation size between two consecutive layers should be at least as large as the size of two messages. Otherwise, the overhead beats the purpose of generalization since a query that is answered by using a higher layer relation can always be answered by using a lower layer relation.

In general, the layers can be constructed based on the heuristics and domain knowledge. It should also be noted that the high layers should be adaptive since user interests and their queries change over time.

4 Intelligent query answering in an MLDB

Queries in MLDBs should be answered using the highest layer relations possible to reduce the communication cost. A query consists of provided information (*query condition*) and inquired information (*query return*), both of which could matching relations in different layers.

To determine the highest layer relations which are able to answer the query, the following steps are needed.

1. For each condition in query condition, decide the highest layer relation in which it can be evaluated.
2. For each attribute in query return, decide the highest layer relation in which it exists.
3. For each relation involved, select its lowest layer from the first two steps.

In the last step, the lowest layer of each relation is selected since any higher layer will not be able to answer the query.

Example 2 Some example queries in the tourist database given in Example 1 are given as follows.

1. Find the name, address, open time, and price of attractions.
This query is best answered using (generalized) relation *attraction*³.
2. Find the name, address, open time, and price of attractions who has a video titled “Water Fun”.
This query is best answered using (generalized) relation *attraction*².
3. Find the name, address, open time, and price of attractions who has a picture of a golden colored object.
This query is best answered using (generalized) relation *attraction*¹. Of course, this assumes that images can be queried by their colors.
4. Find the name, address, open time, and pictures of attractions.
This query can only be answered using (generalized) relation *attraction*.

□

Queries are usually answered by progressively stepping down the layers to find more detailed information. Such a presentation often gives a user better idea on what should be searched further without causing a jam in communication. In such cases, the schema in MLDB acts like a tour guide to locate related lower layer relation(s).

5 Conclusions

A multi-layered database (MLDB) model is proposed and examined in this study, which demonstrates the usefulness of an MLDB in intelligent query answering in mobile databases. The construction of an MLDB and the intelligent query answering in an MLDB are also studied.

In the future, we plan to implement a prototype MLDB system. Another interesting direction is to improve the methods for the construction, maintenance and querying of MLDBs.

References

- [1] E. Gignere. Mobile data management: Challenges of wireless and offline data access. In *Proc. International Conf. on Data Engineering (ICDE)*, pages 227–228, 2001.
- [2] J. Han, Y. Fu, and R. Ng. Cooperative query answering using multiple-layered databases. In *Proc. 2nd Int. Conf. Cooperative Information Systems*, pages 47–58, Toronto, Canada, May 1994.
- [3] J. Han, Y. Huang, N. Cercone, and Y. Fu. Intelligent query answering by knowledge discovery techniques. *IEEE Trans. Knowledge and Data Engineering*, 8:373–390, 1996.
- [4] A. Massari, S. Weissman, and P. K. Chrysanthis. Supporting mobile database access through query by icons. *Distributed and Parallel Databases*, 4:249–269, 1996.
- [5] R.L. Read, D.S. Fussell, and A. Silberschatz. A multi-resolution relational data model. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 139–150, Vancouver, Canada, Aug. 1992.