Journal of Computational Science & Engineering 22 (2016) 680-684

Available at http://www.asocse.org ISSN 1710-4068

Research on Key Technologies of Music Data Management Based on Graph

Yutong Liu*

College of Music and Performing Arts, Yibin University, Yibin, Sichuan 644007, China

Abstract: **Considering the deficiencies of existing music data models and queiy languages, this paper presents a graph-based music data model named Gra-MM to model music data with complicated relationships. The definition of model's logical data structure and algebraic operations are given. Then based on Gra-MM, this paper present a music data query language called Gra-MQL, giving the BNF syntax in the language. Gra-MQL can handle the complicated relationships among music data well, and also has the ability to query music by meta data as well as music content data, which meets the demands of various users and overcomes the shortcomings of traditional graph query languages which do not have adequate expressive power when dealing with data that have complicated relationships and don't have the ability to query music based on music content data.**

Keywords: Music data; Data model; Query language

Received 2 January 2016, Revised 23 January 2016, Accepted 25 January 2016

*Corresponding Author: Yutong Liu, liuyutong123@163.com

1. Introduction

Most existing online music site that offers only retrieval method based on the song metadata, for example, according to the artist's name, song's name or style to query. Music metadata based query methods require music metadata of the song the user want to query. So it increases the difficulty of search. Miscellaneous features and changes are difficult to describe based solely on the metadata. Users may just remember listening to the song who caused melody, but do not understand additional information, therefore it's hard to retrieve the song wanted to be checked in these systems.

In addition, there is a complex relationship between music data. Music management system not only store the audio files and the author, title, time and other music metadata, but also need to store the audio feature vector, pictures, videos, site users' social relations, user reviews of music distribution label and other related data. Between these data more complex association between such two songs have a similar cover other relations; there can be lyricist, composer, singer and other relations between the songs and the singer; the complex relationship between the sites concerned users; the users' favorites, songs' ratings, tagging and other acts between them.

Figure data model is a through vertex, edges and their attributes to represent and store information data model, all data operations on the diagram. Compared with other data model, data model has the following advantages:

(1) Data structure allows us to more naturally to data modeling. In social networks (Social Network), geographic information systems (Geographic Information Systems) and other areas, the complex relationships between data modeling is an extremely important content.

(2) Data query and modification can be conducted on graph data structure, figure lets users become highly abstract operation.

(3) In the implementation process, the map database can use graph algorithms to perform specific operations, such as graph traversal, sub graph matching shortest paths.

(4) The figure database because of its natural characteristics can be more naturally and view the data.

Map data management and query in recent years has become one of the hot areas of data management, complex relationship between music data makes it ideally suited for use graph data model to model. The relationship between music data and data deemed a big picture; a music track can be expressed as a figure vertex, similarity, cover the relationship between tracks and the like can be expressed in music track edge between the vertices. Similarly, the singer also be manifested as vertices in the graph, cooperative relationship between them can be naturally show a side of the singer vertex. User for track collection, scoring, playing tag behavior can be seen as the establishment of edges and vertices between the user track vertices. Music data query is looking for bigger that satisfy a given predicate conditions subgraph.

2. Music Data Model

2.1. Logical data structure

Definition 1. The label: In the Gra-MM, each vertex (edge) with labels, used to represent the vertices (edge) type.

For example, in the music database vertex labels can be *Artist, Track, Band, Album* means, etc., the edge of the label can be *Sing, Cooperate, Similarity*, etc. In this article, we also called vertex (edge) label the vertices (edge) type.

Definition 2. Properties: Property is a binary group (the attribute name, domain), the domain is the set of

2016 © The American Computational Science Society. All rights reserved.

attribute value.

For example, in a music database, has the *Artist* tag vertex has the properties such as *Name, Gender, Track* type vertex has the *Title, Genre* and other metadata attributes used to describe the song. In addition to support content based music data Chad, with songs stored in a binary form for the type of vertex and the content of the *File* and store songs characteristic Feature attribute; Sing of the class type edge *Performance Time, Performance Location* properties, etc. Vertex and edge of attribute information enables the GraMM to express the music data of complex structure and semantics, but also makes the GraMM support based on the content of the music of Chad.

Definition 3. Music database schema: GraMM model is a group of seven elements $S=(N_S,E_S,L_{SN},L_{SE},\lambda_S,A_S,\psi_S)$, in the form of directed labeled graph, said N_S one vertex set; $E_S \in N_S \times N_S$ is the edge set; L_{SN} is a vertex labels; L_{SE} is the edge of the label; $\lambda_S:(N_S \to L_{SN})\cup(E_S \to L_{SE})$ is the distribution of vertices and edges of the label function; A_S is the set of properties which is supported by database; $\psi_S:(L_{SN}\cup L_{SE})\to p(A_S)$ is from tags set to the attributes set function of power set, $p(A_S)$ is A_S in its curtain set (the power set). Through λ_S and ψ_S function, we can know that with the same label (edge) with which attributes to the zenith (Table 1).

In addition, the model *S* will meet the following two constraints:

① $a\in N_S, b\in N_S, a\neq b \Rightarrow \lambda_S(a)\neq\lambda_S(b)$, means the pattern of different vertex must have different labels.

② $a\in E_S, b\in E_S, a=b=(n_1,n_2)\Rightarrow\lambda_S(a)\neq\lambda_S(b)$, means between two vertices in the mode of if there are more than one edge, then these edges must have different labels.

Figure 1 is a simplified music database model. This model defines the music database there are two types of the vertex and two types of edge, and the vertex of the attribute is given. In this paper, the first section 3 to use the method of demonstration to Chad for statements, so gives a example of simplified music database model is easy to understand, the model only defines three types of vertices and five types. In the actual application, defined in the schema will be more than three vertex types.

Table 1 The label and corresponding properties

Label	Properties
Track	Title, Genre, File
Artist	Name, Gender
User	ID
Sing	Performance Time
Similarity	Sim
Like	
Follow	Established Time

Figure 1 is a simplified music database model. This model defines the music database there are two types of the vertex and two types of edge, and the vertex of the attribute is given. In this paper, the first section 3 to use the method of demonstration to Chad for statements, so gives a example of simplified music database model is easy to understand, the model only defines three types of vertices and five types. In the actual application, defined in the schema will be more than three vertex types.

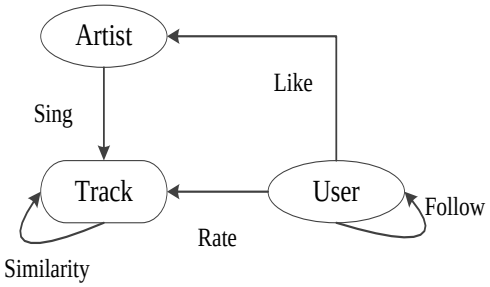

Figure 1. A simplified music database model.

Definition 4. Music database instance: *A* music database meet mode $S=(N_S,E_S,L_{SN},L_{SE},\lambda_S,A_S,\psi_S)$ is a directed labeled graph $G=(N,E,L_N,L_E,\lambda,A,\psi)$, where *N* is a vertex set; $E\in N\times N$ is the edge set; L_N is the culmination of the label set; L_E is the edge of the label; $\lambda:(N\to L_N)\cup(E\to L_E)$ is for the vertices (edges) label distribution function; *A* is the database attribute set support; from the label set to an attribute set power set of functions, which is a screen set (power set) attribute collection; $\varphi: p(A)\to\{dom(a)|a\in p(A)\}$ each $p(A)$ element will map to all the properties of the elements of the Cartesian product (to be described convenience, here referred to $dom(a_1)\times dom(a_2)\times\cdots\times dom(a_k)$ as $dom(a)$, which $dom(a_1)$ is the range of property a_1 ,through λ, ψ and Φ function, you can get the database each vertex (edge) of the property value.

Figure *G* and Mode *S* also exists a subjective function $f: G\to S$, the G vertex (edge) is mapped to the vertex *S* (Side):

① $\forall n\in N(G), \lambda(n)\in dom(\lambda_S(f(n)))$, $dom(a)$ is the label of a domain.

② $\forall e(a,b)\in E(G), f(e)=(f(a),f(b))$ and $\lambda_e\in dom(\lambda_S(f(e)))$.

Figure 2 shows an example and model mapping example. Vertex "Half of My Heart" festival holiday after subjective function mapped to the Track the vertices in the database schema; Vertex "Taylor Swift", "John Mayer" mapping to Artist vertex; The two sides in the instance is mapped to Sing.

Definition 5. Query graph: Query graph is a tuple $Q=(G_Q,\rho)$, which G_Q is an instance of a database schema, ρ_u is a vertex (edge) on u constraints

2016 © The American Computational Science Society. All rights reserved.

property value (predicate), is a series of predicate conjunctions, verbs include =, ≠, ≤, ≥ , <,> and other more relationships.

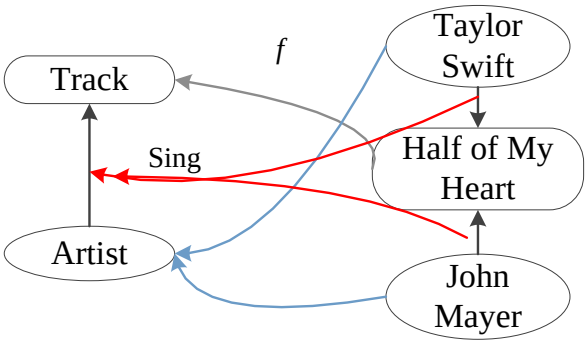

Figure 2. Maps between database instance and modes.

2.2. Algebraic manipulation

A set of algebraic operations is as follows:

Definition 6. Based on the isomorphic subgraph property: Given a graph *G*, a query graph *Q*, said *Q* based property subgraph isomorphic to *G*, if there exists a function such $\phi:(N(Q)\to N(G))\cup(E(Q)\to E(G))$ that the following conditions are true:

①Φ is a single-shot function.

② $\forall(u,v)\in E(Q)$, there is $(\phi(u),\phi(v))\in E(G)$

③ $\forall u\in N(Q),\lambda(u)=\lambda(\phi(u))$, and satisfy the predicate *Q* query graph on $\phi(u)$ property values.

④ $\forall(e)\in E(Q),\lambda(e)=\lambda(\phi(e))$, and satisfy the predicate *Q* query graph on $\phi(e)$ property values.

Referred to as $Q\subseteq {}^{\phi}G$

Definition 7. The query graph match: Given a graph *G*, a query graph *Q*, in the subgraph isomorphism function φ attribute based *Q* matching in *G* is denoted $\phi_Q(G)$.

Figure G may have a plurality of different sub-map and query graph match.

Definition 8. Choice: In GraMM, select the operation input, output figure collection. Given a set of graphs *C*, a query graph *Q*, select Options referred to as:

$$\sigma_Q(C)=\{\phi_Q(c)|c\in C\}$$

Options will return all matching to the sub-graph.

Definition 9. Projection: Projection operation from one set of graphs to select certain vertices, edges on the *C* of each element. Projection operation referred to as:

$$\pi_A(C)=\{c[A]|c\in C\}$$

Which $c[A]$ represents derived from Figure c subgraph. Operating results for the same set of FIG.

Example 1 query "What Taylor cooperation together with the female singer of songs" with algebraic operations can be written as:

$$C=\pi_{trackNode}(\sigma_Q(G))$$

Wherein the query graph *Q* shown in Figure 1, G is a music database instance. This operation is the first operation to find and *Q* by selecting the sub-graph matching the figure from the database, and then use to get songs vertex projection operation.

Definition 10. Portfolio consolidation: Given a set of graphs *C* and a set of graphs *H*, *C* and *H* and referred to as:

$$C\cup H=\{c|c\in C\vee c\in H\}$$

Where *C*, *H* having the same label, and all sessions of the equal value of the vertex (side) will be merged together.

Definition 11. Set difference: Given a set of graphs *C* and a poor set of graphs *H*, *C* and *H* are referred to as:

$$C-H=\{c|c\in C\vee c\notin H\}$$

Definition 12. Set of graphs Cartesian product: Given a set of graphs *C* and a set of graphs *H*, *C* and *H* of the Cartesian product of:

$$C\times H=\{c\cup h|c\in C,\ h\in H\}$$

Definition 13. Music content feature extraction: Feature extraction operation using the specified feature extraction method to obtain songs music content values.

Feature extraction operation referred to as:

$$Extract(N,N_{extract},\xi)=$$

$$\{n_{extract}|n\in N\wedge n_{extract}\in N_{extract}\wedge Feature(n_{extract})=\xi(n)\}$$

Where N is the set of vertices of a song with the same label, which has a type of vertex attributes and attribute values of a store songs feature some song content stored audio format. $n_{extract}$ is used to store the values of the vertices of the n extracted features, $Feature(n_{extract})$ is a characteristic value of the property vertex $n_{extract}$, ξ is the music feature extraction methods.

Definition 14. Match of music content: Content matching operation uses the specified music content matching method to calculate the degree of match between songs. This operation is defined as:

$$Match_o(N,\delta,\gamma)=\{n|n\in N\wedge\delta(o,n)\geq\gamma\}$$

Where o is the given sample song vertex, *N* is set to be matched songs vertices, δ is to calculate the degree of matching method, γ is matching. Matching operation method by δ, the *N* match the degree o of not less than γ song sample song vertex returned.

3. Music Query Language

In this section, the query language GraMQL is introduced based on the model GraMM. GraMQL contains data definition language and data process language.

3.1. Data definition language

First, the BNF **syntax** of the mode creating statement is introduced.

2016 © The American Computational Science Society. All rights reserved.

<create graph statement>::= C RE A T EG R A PH < graph name> <component list>
<component list> ::= <component> [{ <component> }...]
<component>::= <node type definition>|<edge type definition〉
|<extraction methods statement>|<similarity methods statement>
|<similar note methods statement>
<node type definition>::= NODE<node type name><attribute list>
<edge type definition>::= EDGE<node type name><attribute list>
{FROM<node type name>TO<node type name〉 }
<extraction methods statement >:: =
EXTRACTIONMETHODS<methods list>
<feature methods statement > ::=
SIMILARITYMETHODS<methods list >
<similar note methods statement>::= SIMILARNOTE<methods list ><attribute list>::=<attribute item>[{< attribute item >}...]
<methods list>::=<method item>[{<method item >)...]
<attribute item> : : =<attribute type><attribute name>|: ID |
<method item>: : =<method name>[: DEFUALT]

By mode creating statement, GraMQL organizational structure provides the music data in the database, including vertex 1) The type of edge and their attributes appears in the music database; 2) Each type of edge can be connected to vertices; 3) Correlation algorithm could be used for processing music contents. After the schema definition, data manipulation on the model database instance needs to follow the pattern of conduct regulations.

3.2. Data process language

Data process language contains FIND, INSERT, EXTRACT, DELETE, etc. The definitions and the BNF syntaxes of them are as follows:

FIND statement is used to search a database of all the query graph matching subgraphs. Users first need to build a query graph with the edges and vertices, and in which the query criteria is given , and then use this query diagram isomorphic subgraph to query based on attributes in the database.

<find statement>::= FIND<element list>IN<graph name>[<where statement>] [<return statement>]
<element list> ::= <element>[{<element>}...]
<element> ::= <node> I <edge>
<node> ::= NODE<node typo <node name>[<attributes>]
<edge> ::= EDGE<edge type><edge name>[<attributes>]
FROM<node name>TO<node name>
<attributes> ::= <attribute> [{<attribute>)...]
<attribute> ::=<attribute name> = <attribute value〉
<where statement>::=<similarity statement><music note statement〉
<return statement>::=RETURN<element list>

INSERT statement is used to insert data to the database.

<insert statement>::=INSERT<element list> INTO <graph name>

DELETE statement is used to delete data from the database.

<delete statement>::=DELETE<element list>

EXTRACT statement is used to extract feature data.

<extract statement>::=EXTRACT<music blob> INTO <feature blob> [USING<extraction method>]

SIMILARITY statement is used to calculate the degree of similarity between music blobs.

<similarity statement>::=SIMILARITY BETWEEN <music blob> AND < music blob >[USING<extraction method>]

Here is a case:

Insert a new song named “A” to the singer “B”, the song was first sung on October 1, 2015.

```
FIND(
    NodeArtist artist(name='B')
) IN musciDB
INSERT(
    NODE Track newTrack(title='A')
    Edge Sings (performanceTime = '2015.10.01')
        FROM artist TO newTrack
)INTO musicDB
```

3.3. Query processing algorithm

Firstly, the following graph-based music Baseline data query algorithm framework. The Baseline algorithm based on the proposed algorithm Ullmann, adding support for music content pruning operations (row 2, lines 5-7, lines 17-18). Content Pruning algorithm called line 5 is a concrete implementation of the pruning algorithm music content.

In addition to music content is added to the operation in the query algorithms support, we propose to work with the main difference between the two lies in the optimization of existing work methods, in Chapter 3 of this article Query Optimization section will be described in detail

In Algorithm 1, the second line of code to get the query graph vertices search order, as is the Baseline method, the order obtained here both queries in the order defined vertices; 3 lines of code to obtain default music content prune position, namely, music content prune (section 3.2 describes the optimization method) Searching for music content to the candidate point. Function Search (G, Q, S, P, depth, M) , the first row first call Content Pruning (depth, M, p) function to determine whether the current search tree depth required properties according to the music content pruning shears plan branch. Line 11 and line 12, respectively, showing the structure pruning and pruning music meta data candidate point set here will use

2016 © The American Computational Science Society. All rights reserved.

neighbor vertex index and database features available statistical information (tag frequency). The first 13 lines to 20 lines, recursively try to match each candidate point. Line 17, if the current depth than just been pruned tree node depth is larger, then an ancestor of the current node branch has been pruned music content, so from the beginning of this ancestor node sub-tree will be subtracted , thereby reducing the size of the search space. It should be noted that, due to the method in Algorithm 1, music contents pruning positions not optimized, so the first 17 lines, 18 lines of code will not be executed to.

Algorithm 1: Music Subgraph Querying
Input：Data graph *G*，Query graph *Q*
Output：Subgraphs in *G* that are isomorphic to *Q*
Boolean: this brunch can be pruned
1 *S*←GetSearchOrder(*G*,*Q*)//to be optimized
2 *P*←GetContentPruningPlan(S) // to be optimized
3 *depth*←0， *M*←∅
4 **Search**(*G*, *Q*, *S*, *P*, *depth*, *M*)
5 **if**ContentPruning(*P*, *depth*, *M*) **then**
6 return//this brunch should be pruned
7 **endif**
8 **if** depth=|*N*(*Q*)| **then**
9 **report** *M* and **returntrue**
10 n←S.nextQueryNode, *Can*←getCandidate(n)
11 refine *Can* by structure pruning
12 refine *Can* by meta pruning
13 for each *m*∈*Can***do**
14 add *m* to *M*
15 **Search**(*G*, *Q*, *S*, *P*, *depth*+1, *M*)
16 remove *m* from *M*
17 if depth >*P*.justPrunedDepth **do**
18 **return**
19 **end if**
20 **end for**
21 **return**

Algorithm 2: ContentPruning
Input: Search tree *depth*, Matched nodes Content pruning plan *P*
Output: Nodes that should be pruned by content
Boolean: this brunch can be pruned
1 *indexes* ←*P.* nodesToBeCheckAt(*depth*)
2 ifindexes ≠ ∅ then
3 for each *i*∈*indexes* do
4 n_{con}←*M*[i]
5 if content op has not been done on n_{con} then
6 do content op on n_{con}
7 end if
8 if n_{con} can be pruned by content then
9 *P*.justPrunedDepth← *depth*-1
10 return true
11 end if
12 end for
13 end if
14 return false

Algorithm 2 is a music content pruning algorithm execution, the process is as follows: First Line 1-2 determines whether the current search tree depth required music content prune plans, if any, need to perform music from the get all the content in the content pruning program P candidate vertex index operations. Then the cycle is determined whether the content needs to be pruned every vertex has been defined predicate music content operations (line 5> If not, the operation is performed. Line 8 to determine whether the vertex music content can be pruned, if Yes, the current record prune depth and then return to inform top-level query algorithms.

4. Conclusion

A graph-based music data model named Gra-MM to model music data with complicated relationships is presented in this paper. The definition of model's logical data structure and algebraic operations are given. Then based on Gra-MM, this paper present a music data query language called Gra-MQL, giving the BNF syntax in the language. Gra-MQL can handle the complicated relationships among music data well, and also has the ability to query music by meta data as well as music content data, which meets the demands of various users and overcomes the shortcomings of traditional graph query languages which do not have adequate expressive power when dealing with data that have complicated relationships and don't have the ability to query music based on music content data.

References

[1] L. Chen, C. Wang. Transactions on Knowledge and Data Engineering, 22 (2010) 1093-1109.
[2] M. Potamias, F. Bonchi, A. Gionis. Proceedings of the VLDB Endowment, 3 (2010) 997-1008.
[3] Y. Yuan, G. Wang, H. Wang. International Conference on Very Large Data Bases, 2011.
[4] H. Shang, Y. Zhang, X. Lin. Proceedings of the VLDB Endowment, 1 (2008) 364-375.
[5] L.P. Cordella, P. Foggia, C. Sansone. Transactions on Pattern Analysis and Machine Intelligence, 26 (2004) 1367-1372.

2016 © The American Computational Science Society. All rights reserved.