# THE ANATOMY OF A BIBLIOGRAPHIC SEARCH SYSTEM FOR MUSIC

*Ryan Scherle*
Indiana University
Digital Library Program
rscherle@indiana.edu

*Donald Byrd*
Indiana University
Digital Library Program
donbyrd@indiana.edu

**ABSTRACT**

Traditional library catalog systems have been extremely effective in providing access to collections of books, films, and other material. However, they have many limitations when it comes to finding musical information, which has significantly different, and in many ways more complex, structure. The Variations2 search system is an alternative system, designed specifically to aid users in searching for music. It leverages a rich set of bibliographic data records, expressing relationships between creators of music and their creations. These records enable musicians to search for music using familiar terms and relationships, rather than trying to decipher the methods libraries typically use to organize musical items. This paper describes the design and implementation of the system that makes these searches possible.

## 1. INTRODUCTION

While searching collections of musical works by content is an interesting problem, and has been studied by the Music Information Retrieval community at length [11], bibliographic searching is equally important. It is also, perhaps, equally challenging: the exceptional demands of music are well-known to catalogers, and they have been studied in detail by library scientists [13].

Variations2 is a digital library system designed to improve access to and use of music. The Variations2 search system focuses on improving bibliographic searches of musical materials. Searching for music, either by content or by metadata (bibliographic information), is a complex problem for several reasons:

1. **Multiple versions:** Famous books may be available in a variety of languages, but it is rare for a library to contain multiple versions of a book that would be of interest to anyone other than scholars (the Bible being a notable exception). Users may be interested in finding a particular translation of *Canterbury Tales*, but often as not, they simply want to read the story. With music, the library is more likely to have multiple versions, and users are more likely to prefer one version over another (e.g. Bing Crosby's "White Christmas", rather than a version by Elvis Presley, The Beach Boys, or Garth Brooks).
2. **The whole/part problem:** Musical works often exist in a hierarchical structure. For example, an opera has a name, and it contains many named arias. A user may be interested in the work as a whole, or simply in one of the arias.
3. **Related works/variations:** A portion of a book or film may be quoted in another work of the same type, but the relationship is usually very straightforward. Music, however, is often copied, varied, and reused in unexpected ways, like the appearance of the "Ode to Joy" theme from Beethoven's 9th Symphony as a major theme in the movie "Die Hard" or the excerpt from *La Marseillaise* at the beginning of the Beatles' song "All You Need Is Love".
4. **Instrumentation:** Music is a performing art, one that is performed by an enormous variety of instruments as well as voices, and the performing media are often important to a searcher. Instrumentation also greatly increases the importance of multiple versions. Pianists play arrangements of pieces originally written for violin, violinists play arrangements of songs, etc.
5. **Language:** A problem specific to metadata searches is that of language. Someone searching for Proust's "Remembrance of Things Past" probably would not be interested in items cataloged under the original title, "À la recherche du temps perdu"; they almost certainly would not be interested in all versions of the work, regardless of language. However, it's very likely that anyone who asks for the "Rite of Spring"–the famous ballet by Stravinsky–would be equally interested in "Le Sacre du Printemps", since the title of a publication of music says almost nothing about the content.

Digital library projects that store music typically mimic standard library practice, storing a single record for each item (e.g. CD, LP, score) in the collection, with no explicit relationship between items. One such system is the predecessor of Variations2, Indiana University's highly successful VARIATIONS project [5]. VARIATIONS provides links to digital media directly from records in the library's

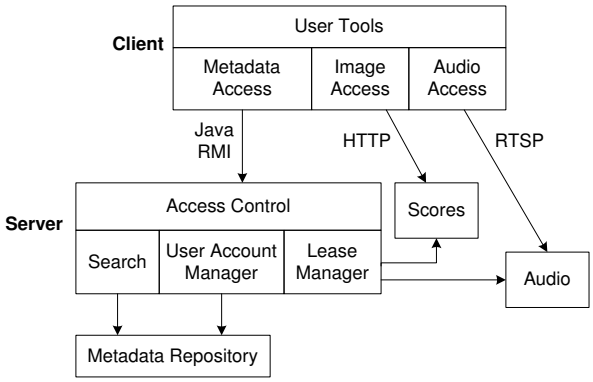


**Figure 1**. System architecture

online catalog. Since the search functionality is handled by the standard library catalog, the problems described above affect search results, often making music difficult to find.

Smiraglia [12] argues that to correct these problems, music retrieval systems must be centered on the musical work, rather than the individual item in a library's collection. The Functional Requirements for Bibliographic Records (FRBR) developed by the International Federation of Library Associations and Institutions [7] makes a similar argument for general library materials, and some FRBR-based systems have begun to appear.

To date, few music projects have adopted work-based models. One system that attempts to manage work information independently of album information is the All Music Guide [1]. This system provides links between albums and works, allowing for some navigation between versions of a work, but the system is still largely item-based.

Variations2 uses a work-based model to support searching and browsing. This model collects all versions of a work in response to searches, regardless of instrumentation or language. Links are provided from whole works to their parts, and searches for a named portion of a work will find the relevant whole work. The next section gives an overview of the Variations2 architecture, which provides a framework for the subsequent description of the search system itself.

## 2. SYSTEM ARCHITECTURE

Variations2 allows users to access music in a variety of formats (audio recording, score images, and, soon, encoded scores). The Variations2 system has a modular design, separating storage of media files from storage of the bibliographic metadata. The structure of the system is shown in Figure 1.

Users primarily interact with the Variations2 client. The client contains low-level modules that access metadata and play/display the library's media content. User-level tools, built on top of these low-level modules, include an audio player and a score viewer, as well as a tool for synchronizing playback of recordings and scores, and a tool for diagramming the form of a musical piece. The client is written in Java, using the Swing user interface toolkit, which makes it portable. The full version of the client runs on Windows and Mac OS X, while many non-graphical functions can also be performed on other operating systems, including Linux and AIX.

Clients use Java's protocol for Remote Method Invocation (RMI) to communicate with the Variations2 server. The server mediates access to all library materials, including audio recordings, scanned score images, and the repository of metadata describing these materials. All requests to the server must pass through an access-control system. Like the client, the server is written in Java, and can run on multiple operating systems.

Bibliographic records and user account information are stored in a metadata repository using IBM's relational database system, DB2. To facilitate searching of textual information, the database uses DB2 Net Search Extender. The database design is described further in section 5.2.

After a search has identified bibliographic records for desired media items, users can open the items in the appropriate tool. When the client requests access to an audio recording, the server checks that the user is allowed to access the recording, and then issues the client a *lease*. The lease contains a URL to media on the streaming server. This URL is valid for a limited time period, forcing the client to renew the lease periodically. The streaming server runs Apple's Darwin Streaming Server package. Using QuickTime for Java, the client has full control over any media for which it holds a lease. Audio is streamed in two formats: 192kb bps MP3 for high-bandwidth connections and 32kb bps AAC format (within MPEG4) for low-bandwidth connections.

A similar process is followed when the client requests a score image. In this case, the lease contains the URL of a multi-page image on a Web server. Score images are stored in the DjVu format developed by AT&T Labs [4]. DjVu's compression technique, which separates foreground information from background information before compressing, allows for files of small size to retain extremely high quality. Other compression formats, which try to compress foreground and background together, tend to lose significant musical markings (e.g. note stems, dotted notes) and distort staff lines.

## 3. THE SEARCH PROCESS

From a user's point of view, searching Variations2 is similar to using a search engine on the Web. The basic search screen, shown in Figure 2, allows the user to enter a creator/composer of a musical work, a performer, the name of the work itself, the work's musical key, or the type of media on which the work appears. Most of the search results are hyperlinks which lead to more detail and/or media. Users can also page forwards and backwards through previously-viewed result screens.

A music student searching for works by J. S. Bach may simply enter "bach" into the creator/composer field, as

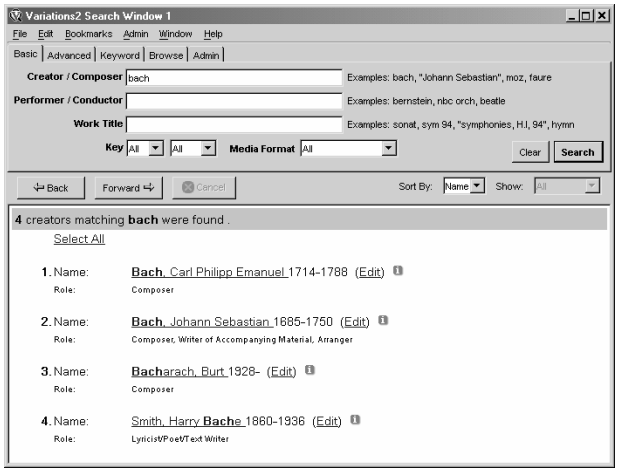


**Figure 2**. Search screen

shown in Figure 2. Results are displayed in the bottom half of the search window, indicating that four creators in the system match the name "bach".

In this case, the user is not immediately presented with a list of library holdings (i.e. sound files and scores). Presenting a complete list would cause problems similar to those of standard library search systems. Instead, the user is asked to disambiguate the query by choosing one of the creators matching "bach". Clicking on the entry for "Bach, Johann Sebastian" will retrieve a list of the works composed by him that are available in the system.

Throughout the disambiguation process, if a result set contains exactly one item, that item is automatically selected, and the query continues without user intervention. For example, if the user searches for creator name "beethoven", and there is only one result, the client will immediately issue a new query to get Beethoven's works, and the first result screen shown to the user is the list of works. This allows users to provide only as much information as is necessary to retrieve the desired content. If, during the disambiguation process, the user prefers to view all matching items, they may click on the "Select All" link at the top of the results (see Figure 3).


**Figure 3**. Search results detail

The number of disambiguation screens a user must go through is dependent on the query fields the user completes as well as the contents of the library, but is never more than four, and is typically only one or two. When a user reaches the end of the disambiguation process, they will reach a screen that displays occurrences of the requested work in the library's collection. Figure 4 shows the results of selecting "Bach, Johann Sebastian" from the creator listing, and then selecting "Brandenburgische Konzerte. Nr. 1" from the list of works. Variations2 currently includes three occurrences of this work, two recordings and a score. The result screen includes buttons to open each item in the appropriate media player.

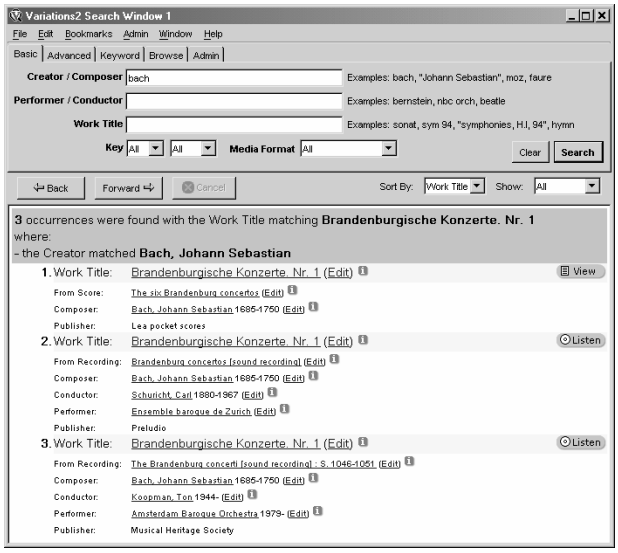


**Figure 4**. Search results: library holdings

In addition to the basic searches described above, users can click one of the tabs at the top of the main search window (see Figure 2) to access additional searching options. An advanced search screen provides more query fields than the basic screen, including subject headings, album/score title (as opposed to the title of a single work), and contributors other than creators or performers (described below). A keyword search screen provides searches more typical of existing library search systems, allowing search over terms appearing anywhere in relation to a recording or score. A browsing system allows users to quickly learn the extent of the system's content. Catalogers and system administrators may use an additional search screen that supports search over administrative data, including the current status of a record and the time it was created or last modified.

## 4. THE DATA MODEL

To search information in this way, the metadata must be structured to accurately reflect the relationships between musical entities. The MAchine-Readable Cataloging (MARC) [9] records that are standard in library catalogs have many difficulties capturing these relationships [6].

The Variations2 data model, shown in Figure 5, is an entity-relationship model that allows for rich representation of the relationships between musical works, creators, performers, and performances, with a minimum of redundant information. This model is very similar to the FRBR model [7]. The Variations2 data model includes five record types:

- **Work:** Represents the abstract concept of a musical piece or set of pieces.

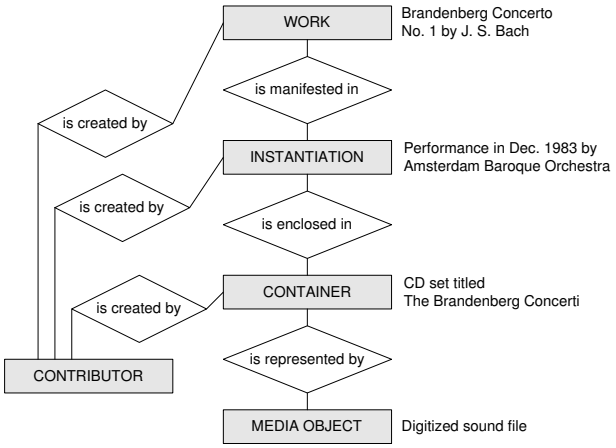


**Figure 5**. Data model example

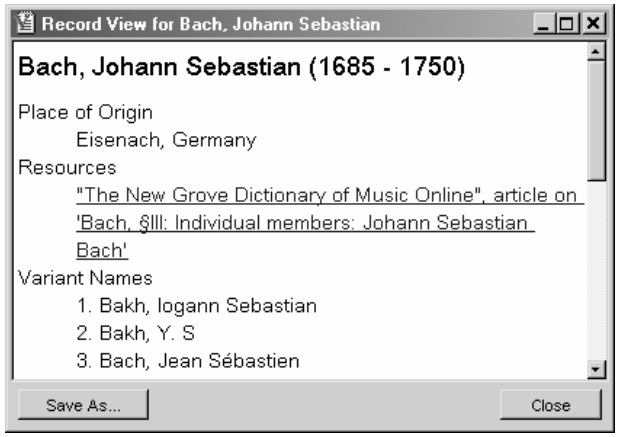


**Figure 6**. Details for Contributor J. S. Bach

- **Instantiation:** Represents a manifestation of a Work as a recorded performance or a score.
- **Container:** Represents the physical item (or set of items) on which Instantiations can be found. There may be many Instantiations on a single Container.
- **Media Object:** Represents a piece of digital media content, such as a sound file or score image.
- **Contributor:** Represents a person or group who contributed to the creation of music.

To facilitate searching, Contributors are broken down into three classes based on the nature of their contribution:

- **Creator:** A Contributor attached to a Work. That is, a person/group who contributed directly to the creation of the musical piece, but not necessarily to any particular performance or score. This is usually a composer or lyricist.
- **Performer:** A Contributor attached to an Instantiation. This is a person/group who contributed to the creation of a particular performance or publication of a Work. This usually corresponds to the traditional definition of a performer, but can also encompass contributors like conductors and remix artists.
- **Other Contributor:** A Contributor attached to a Container. This is a person/group who contributed to the creation of a the actual album or score volume. While this designation is not used very often, it can be useful for famous producers, or individuals who write liner notes.

Each type of record holds a variety of fields containing descriptive, administrative, and structural metadata. Figure 6 shows the Variations2 details screen for J. S. Bach, illustrating some of the descriptive information that is stored in Contributor records. Administrative metadata includes information about the record's creation, the number of times it has been accessed, and information about related records in other library systems (e.g. OCLC or IU's main library catalog). Structural metadata includes both information about relationships between the records (as shown in Figure 5) and structure within a record. A Container record representing a set of CDs will have a Container Structure with one node for each CD, and subnodes for each track on a CD. Likewise, the structure of a Container for a score will have a hierarchy of nodes, with the lowest level nodes representing individual pages. This structure can be used by the various user tools to navigate the Container.

Usually, a Container record will contain structure appropriate for navigation, but in some cases, the structure of a Container does not correspond to Works in the system. A user may be interested in hearing The Beatles perform "Blue Suede Shoes". However, the only occurrence of this performance in the library is part of a medley, and begins in the middle of a CD track. To overcome this problem, Variations2 uses a structural metadata element called a Binding, which connects an Instantiation, Work, and Media Object. The Instantiation for The Beatles' performance of "Blue Suede Shoes" contains specific time offsets into the appropriate Media Object. When the user selects this Instantiation from the search results, the audio player uses the Binding structure to open the media at the correct point. More detailed Bindings can be created as well, down to the level of one Binding for each measure in a piece. The user tools can use this structure to navigate a Media Object with respect to the structure of a Work, independent of the structure represented in the Container. Details about the Variations2 data model can be found in [10] and [8].

## 5. DATA CREATION AND STORAGE

With a highly-connected data model, care must be taken to create and store the records in a manner that preserves the relationships. The records are created by importing data from other sources, augmented with manual cataloging. For storage, the data records are mapped into the tables of a relational database.

### 5.1. Record creation

When a new item (i.e. Container) is added to the system, initial metadata is imported from a MARC bibliographic

record in the library's online catalog system. The import system identifies as much information as possible from the MARC record, and fills the corresponding fields of a Container record. The MARC record for a Container often contains the names of Contributors or Works that appear on the Container, but these names are not explicitly linked to the structure of the Container. A human cataloger uses this information to create new Contributor or Work records, or link to existing ones.

In addition to the MARC bibliographic records used to populate fields in the Container, Variations2 makes use of existing MARC authority records representing many musical works and contributors. These records provide another source from which information may be imported, but some data must still be entered manually. In particular, nearly all information in the Instantiation record (including Bindings) must be generated by a human cataloger, as there is no corresponding information in the MARC format.

### 5.2. Metadata repository

Once records have been created, they are stored in a relational database using IBM's DB2, version 8.1. One table is stored for each record type, as well as a single table to hold information common to all record types (status, creation time, etc.). A set of utility tables stores fields that may be repeated within a record, like notes. Fields such as personal names and album titles that need the advanced searching features provided by DB2 Net Search Extender are copied to a separate index table, called TextIndex.

| **Table Name** | **Contains** |
|---|---|
| Entity | Fields that are common to all primary records |
| Work<br>Instantiation<br>Container<br>Media Object<br>Contributor | Non-repeatable fields of a record, and references to tables with repeatable fields |
| ContributionSequence<br>NoteSequence<br>ResourceSequence<br>⋮ | Repeatable fields, which may come from any of the primary record types |
| TextIndex | Copies of fields that need to be searchable as text |
| UserProfile<br>Group<br>ReserveList | Information about user accounts and their associated privileges |

**Table 1**. Partial listing of database tables

The Contributor record for "Bach, Johann Sebastian", partially shown in Figure 6, is stored in the Contributor table. Each field of the record corresponds to a field of this table. The primary name is copied to the TextIndex table, along with the variant names. The Work table contains a record for "Brandenburgische Konzerte. Nr. 1". This Work record references a set of records in the ContributionSequence table. In this case, the set contains a single record, which refers to the "Bach" Contributor record.

When a record is requested from the database, the fields are retrieved from all tables related to the record, and assembled into a single Java object. Records may be retrieved from the database individually, or as sets in response to searches, as described in the next section.

## 6. SEARCH SYSTEM INTERNALS

The actual work of searching is shared by the client and server. The client generates queries based on the user's requests, and renders results in an appropriate manner. The server takes queries and collects the matching records from the database.

### 6.1. Client search process

When the user clicks the "Search" button, the Variations2 client packages the requested fields into a query for submission to the server. The "bach" query described in section 3 is translated into a structure that looks like this:

| | |
|---|---|
| Constraints | Creator(name) = bach |
| Result context | Creator |
| Projections | Creator(dates, names),<br>Work(contributions) |

**Table 2**. Initial query for Creator "bach"

*Constraints* are a straightforward encoding of the search terms entered by the user, labeled with the corresponding record types and fields. The *result context* indicates the primary type of record being sought at this point in the search process. This is an optional portion of the query, necessary for avoiding conflicts in certain cases. It will be further explained in Section 6.2. A set of *projections* is automatically selected by the client based on the constraints, indicating the types of records and fields that the server should return.

Even though the primary purpose of this query is to return Contributor records, the projection requests Work records as well. This is necessary because the Work record stores information about the role a Contributor played in the Work's creation, and the client will be displaying these roles to aid the user in identifying the correct Contributor. In Figures 2 and 3, roles are displayed under each Contributor's name.

After the query is processed by the server, the client receives a set of records encoded as Java objects. At this point, the record set will be transformed into HTML for display as a disambiguation screen, or if there is only one Creator in the result set, it will be automatically selected. In either case, a Creator will be selected, and a new query will be sent to the server. If J. S. Bach is selected, the new query will look like:

This query requests all Work records that have a Creator with ID 1015 (Bach's ID number). A projection for Creator is also set, since Creator's names are helpful in

| Constraints | Creator(id) = 1015 |
|---|---|
| Result context | Work |
| Projections | Work(all)<br>Creator(primary name, dates) |

**Table 3**. Continued query for Bach's works

the identification of Works, and are always displayed on a Work result screen. When the server returns the result set for this query, the user will be shown the Work disambiguation screen, and the process begins again, until a screen of library holdings (Instantiations or Containers) is reached.

### 6.2. Server search process

The Variations2 server takes queries posed by the client in the format described above, and returns the appropriate Java objects. When the server receives a query, the following basic process is executed:

1. An SQL query is executed to retrieve ID numbers for records of the result context type.
2. An additional SQL query is executed for each projection, retrieving ID numbers of records related to the objects identified in step 2.
3. For each of the ID numbers identified in steps 1 and 2, a Java object is assembled from the database, projecting to include only the requested fields.

Queries may contain any combination of constraints, result context, and projections. Due to this complexity, SQL queries to identify matching records in the metadata repository are not hard coded. Instead, the server generates SQL queries automatically.

All constraints are used at every step of the search, since they are necessary to identify matching records. For example, if the user searched for Creator "bach" and Work title "west side story", the Creator constraint on its own would match several records, but none of these would correspond with records matching the Work constraint.

To ensure queries with multiple constraints correctly traverse relationships between the database tables, the server references a directed graph that stores the names of tables and fields that connect them. This graph is similar to Figure 5, with the addition of nodes to express the definitions of three Creator types described in Section 4. The server performs a depth-first traversal of the graph, starting at the node representing the target data type. When it reaches a node in the graph corresponding to a constraint in the query, it adds the constraint to the SQL statement, also adding select statements that link back to the starting data type.

These SQL queries can get quite long. The query shown in Figure 7 is one of the simplest, requesting Contributors that match the Creator name "bach". The first half of the query searches for the the name "bach" in any TextIndex records connected to the Contributor.name field. The second half of the query ensures that only Creator records are found by linking Contributor records with records in the Work table.

Some constraints refer to fields in the Entity table, and apply to all record types. For queries containing these constraints, an initial SQL query is executed using *only* the Entity constraints, and the results are placed in a temporary database table. The original Entity constraints are then replaced with a constraint indicating membership in the temporary table, and the main query is executed. This allows for more efficient processing than applying the Entity constraints to each table as it is encountered.

Many queries do not require a result context, as the appropriate results could be generated by simply retrieving all records specified in the projections that conform to the constraints. A result context is necessary, however, when requesting the list of Works performed by a Performer. In this case, the constraints require Contributor records to be interpreted as Performers, but we also want to project for Creators, since the Creators are often the primary method for identifying a Work. If the same constraints were applied to all projections, no Creators would match unless they happened to be listed as both a Creator and Performer of the piece.

When the server has completed the process of finding and building matching records, the set is returned to the client for display to the user or use within one of the user tools.

### 6.3. Increasing recall

Because we are using a database for storage and retrieval of the data, the retrieval process finds only records that match the query exactly; partially-matching records are not found. In information retrieval terms, this is 100% precision (every document found is a desired document), but less than 100% recall (not every desired document is retrieved).

Since the contents of the database are focused on music, rather than on a broader collection of topics, precision is less of an issue than would normally be the case in an information retrieval system. Users of a focused system like this often want higher recall. Even though a database is being used, it is possible to trade precision for greater recall. Several techniques are used to increase the number of records matching a query:

- Variations in capitalization, diacritics, and punctuation are ignored. Terms in the TextIndex table are normalized, with punctuation and diacritics removed, and all letters converted to lower case. The same process is applied to any query terms that reference fields in the TextIndex.
- Terms are matched as individual keywords, not as phrases, making the order of terms within a field irrelevant, unless the user explicitly indicates a phrase with quotation marks.
- In the SQL statement, a wild card character is appended to each query term, allowing users to type

```
SELECT Contributor.sql_id FROM TextIndex, Contributor WHERE Contributor.name=TextIndex.sql_id
AND CONTAINS(TextIndex.text, '"bach%"')=1 AND Contributor.sql_id IN
(SELECT ContributionSequence.e_contributorRef_sql_id FROM ContributionSequence, Work WHERE
Work.contributions=ContributionSequence.sql_id)
```

**Figure 7**. SQL statement to identify "bach" as a Creator

only the first few letters of a name or title. This character appears as a percent sign in Figure 7.

- As with MARC authority records, the names of Contributors and are often supplemented with variant names, indicating alternate spellings of, or aliases for, the primary name. Figure 6 indicates some of the variant names stored with the Bach record.

As a result of these techniques, queries as varied as "bach, j. s.", "Johann Bach", and "Iog Seb. Bakh" will match the record for "Bach, Johann Sebastian".

### 6.4. Performance concerns

Although search speed is not a major focus of the Variations2 project, results must be presented to the users quickly enough that they do not become dissatisfied. Executing complex SQL statements, building objects from the database, transferring objects across a network connection, and rendering result screens takes time. Variations2 makes extensive use of caching to quickly provide results for popular searches. Like a Web browser, the Variations2 client caches the results of searches, so the user can page forward and backwards without waiting for the server to answer a query. The server maintains two separate caches. One stores ID numbers that match recently executed queries, so that the database will only be searched once when multiple users execute the same query, as happens often for classroom assignments. The second server cache stores Java objects for recently requested ID numbers, allowing objects to be quickly projected and returned, without excess time building the fields of each object from the database.

In addition to caching results, the size of result sets is limited. At the time of this writing, the Variations2 system contains 1743 Works on 289 Containers. Eventually, we plan to include the nearly 9000 Containers represented in the original VARIATIONS collection [5], and continue growing in response to the needs of the IU School of Music. Non-specific queries that result in large result sets can cause many types of problems, including slowing response times, overflowing the Java RMI connection, or exceeding the amount of memory allotted to the client. Due to the highly connected nature of the data, splitting result sets into multiple pages is problematic, so the system currently places a hard limit on the number of objects in the result context that can be returned at a time.

### 6.5. Avoiding blind alleys

It is possible for queries to result in a "blind alley", where the constraints in the query lead to valid Contributor and/or Work records, but these records either have no associated Instantiations, or the associated Instantiations are not viewable by normal users. These sorts of problems are usually the result of items that are in the process of being cataloged. There are two possible solutions:

1. The server adds default constraints to every search, ensuring that the items found are connected by user-accessible records to valid Media Objects.
2. Cataloging work is performed in a separate database, and records are only copied to the production database when they would meet the criteria in solution 1.

Variations2 currently takes the first approach, but the second approach is being considered as a way to improve performance.

### 6.6. Keyword searches

In addition to the field-based search described above, Variations2 provides a "keyword search" system that allows users to search for terms appearing anywhere in relation to a library item (i.e. a Container). This is similar to the type of search users traditionally perform in an online library catalog.

Keyword search works a bit differently than the basic search, since keywords for a given Container must be drawn from many types of records. For example, users searching for keyword "beatles" would expect to find all Containers that contain Instantiations performed by a Contributor named "The Beatles", and not just Containers that have this term in one of the Container fields.

A KeywordIndex table in the database is used to collect keywords for each Container. This table is initialized with Container titles, so minimal searching can be done when a Container is first entered. Periodically, a process runs to compute keywords and add them to the KeywordIndex. This process collects all Works, Instantiations, Performers, Creators, and Other Contributors associated with a given Container. All fields of these records are added to the list of keywords, except for certain administrative note fields.

During search, the KeywordIndex table functions much like the TextIndex table. Keywords may be used as a constraint with other search fields, and the generated SQL statements use DB2 Net Search Extender to provide string-matching functionality. Since keywords are based on the contents of a Container, only Container records are displayed in the result screen for keyword searches.

## 7. CONCLUSIONS AND FUTURE DIRECTIONS

The Variations2 system has been in use at Indiana University and several satellite sites for two years. The search system has been evaluated in usability tests and in actual library use. Users had no difficulty understanding the search process, and were pleased with both the organization of the data and the search interface.

Cataloging items for use with the Variations2 system requires considerable human effort. This effort is in addition to the effort typically required by the library to catalog a item in MARC format. We are investigating methods for increasing the amount of metadata that can be collected automatically. This includes information outside of MARC records, but also information currently available in MARC records that cannot easily be imported without human intervention. Another solution under investigation is cooperative cataloging, using a methods similar to the manner in which OCLC manages cooperative cataloging for MARC records.

Ultimately, systems that combine bibliographic and content searching are likely to be more effective than either method used alone. Users in search of music often query with a combination of bibliographic and content-based information [2]. As a first test of combined searching, we have integrated Variations2 with the University of Michigan's VocalSearch system [3]. In the combined system, users are able to sing a query for a restricted set of library materials (Beatles songs). VocalSearch identifies songs with matching themes, and forwards the ID numbers of these songs to Variations2. The Variations2 client then requests the matching Works from the server, and presents them to the user in ranked order. This system gives the user the option to begin with a content-based query and use bibliographic information to refine results, retaining the ability to see all available Instantiations (performances or scores) of the Work. Options for content-based searching will be expanded in future versions. Plans are underway to add musical themes to the system, as well as complete musical scores in symbolic form. Both themes and symbolic scores will be searchable in future versions of Variations2.

## 8. ACKNOWLEDGMENTS

We wish to thank the members of the Variations2 development team, who contributed to the design and implementation of the searching system and provided feedback on early versions of this paper. They include Jon Dunn, Mark Notess, George Yang, Jim Halliday, and Rob Pendleton. This material is based upon work supported by the National Science Foundation under grant 9909068. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## 9. REFERENCES

[1] All Music Guide Web site. http://www.allmusic.com.

[2] D. Bainbridge, S. J. Cunningham, and J. S. Downie. How people describe their music information needs: A grounded theory analysis of music queries. In *Proceedings of the Fourth International Conference on Music Information Retrieval (ISMIR 2003)*, Baltimore, MD, October 2003.

[3] W. P. Birmingham, K. O'Malley, J. W. Dunn, and R. Scherle. V2V: A second variation on query-by-humming. In *Proceedings of the Third Joint Conference on Digital Libraries (JCDL2003)*, page 380, Houston, TX, May 2003. IEEE Computer Society.

[4] L. Bottou, P. Haffner, P. Howard, P. Simard, Y. Bengio, and Y. L. Cun. Browsing through high quality document images with DjVu. In *Proceedings of IEEE Advances in Digital Libraries*. IEEE, 1998.

[5] J. W. Dunn and C. A. Mayer. VARIATIONS: a digital music library system at Indiana University. In *Proceedings of the Fourth ACM Conference on Digital Libraries*, Berkeley, CA, 1999.

[6] H. Hemmasi. Why not MARC? In *Proceedings of the Third International Conference on Music Information Retrieval (ISMIR 2002)*, pages 242–248, Paris, France, October 2002.

[7] IFLA Study Group on the Functional Requirements for Bibliographic Records. *Functional Requirements for Bibliographic Records*. K. G. Saur, Munich, 1998.

[8] IU Digital Music Library Data Model Specification. http://variations2.indiana.edu/pdf/DML-DataModel-V2.pdf.

[9] Library of Congress MARC Standards Web site. http://www.loc.gov/marc/.

[10] N. Minibayeva and J. W. Dunn. A digital library data model for music. In *Proceedings of the Second Joint Conference on Digital Libraries (JCDL2002)*, pages 154–155, Portland, OR, July 2002. IEEE Computer Society.

[11] Music Information Retrieval Web site. http://www.music-ir.org.

[12] R. P. Smiraglia. Musical works as information retrieval entities: Epistemological perspectives. In *Proceedings of the Second International Conference on Music Information Retrieval (ISMIR 2001)*, pages 85–92, Bloomington, IN, October 2001.

[13] S. Vellucci. *Bibliographic Relationships in Music Catalogs*. Scarecrow Press, Lanham, MD, 1997.