EMERSE

Electronic Medical Record Search Engine

Implementation and Customization
Technical Manual

project-emerse.org

David Hanauer <hanauer@umich.edu>

Last updated June 16, 2017

Intended Audience

The intended audience for this Technical Manual are developers and IT/informatics support teams with deep technical expertise. The manual covers many details related to installing and customizing the system for a local implementation. This manual does not cover general usage of EMERSE and it is not mean to be a help document for users. Such user-focused help can be found elsewhere. In addition to this document we also offer a separate manual related to our fully functional virtual machine (VM), where all components have already been installed and configured. The VM is a valuable companion to this document since IT teams can use the VM to look “under the hood” and better understand how EMERSE actually works.

License

EMERSE is being offered at not cost under a __________ license.

Warranty? Disclaimers?

Feedback is welcome

This Technical Manual is a work-in-progress. We welcome any feedback to improve it, including corrections, additions, and other clarifications. If at any point you get stuck or run into problems, please contact us so we can help. The main point of contact is David Hanauer <hanauer@umich.edu>.

Partial to do list for this manual:

* Discuss synonyms and how to add/update

* Discuss authenticating users with LDAP or others

* How to add users

* How to delete users

* Sys admin features, if any

* Queries to check on system stats

* Details about Solr indexing

Background

EMERSE is the Electronic Medical Record Search Engine. It was designed to work with the free text (unstructured) clinical documents in an electronic health record (EHR) system. Most importantly, EMERSE was made for regular users, not those with IT or informatics expertise. EMESE has some similarities with natural language processing (NLP) tools, but it is technically an information retrieval tool and it is not what one might traditionally consider NLP software.

EMERSE provides features to help users get their work done quickly and accurately. For example, the tool can provide many suggestions related to synonyms and related keywords, acronym expansions, and even suggestions for both generic and brand drug names.

EMERSE offers intuitive and novel ways to visualize the search results, helping users focus on the information they need. For example, search results can be viewed as a 'heat map' overview, showing the density of documents with a search hit, as well as what we call a 'mosaic view' which shows a color-coded grid with specific terms that appear for each patient in the search results.

EMERSE was developed at the University of Michigan in 2005 and has been continuously improved and updated since then. EMERSE has been used to support many real world tasks including clinical and translational research (both cohort identification and data abstraction), quality improvement and quality assurance initiatives, as well as hospital operational support tasks. It has been used very successfully by the billing and coding team for complex case reviews, improving reimbursement rates by nearly $1 million per year. The software is used routinely by the Michigan Medicine Compliance Office, Office of Risk Management, and the Department of Infection Control. EMERSE has also been used in the clinical setting to support rapid data retrieval to answer clinical questions as well as to reduce the burden of prior authorizations, required by many insurance companies. EMERSE has also been used to support hundreds of research projects, leading to many peer-reviewed publications.

EMERSE can integrate documents from multiple sources. At the University of Michigan, EMERSE contains over 100 million documents, including those from our original, homegrown EHR system, CareWeb, as well as documents from our pathology system, and even documents from our replacement vendor EHR, Epic. EMERSE was designed to be vendor-neutral -- as long as you can get your documents out of the source system, EMERSE should be able to search through them. We've also created an interface to move cohorts identified through the i2b2 Workbench directly into EMERSE for further searching of the patients’ free text notes.

EMERSE was created with support from the University of Michigan Comprehensive Cancer Center, the CTSA-supported Michigan Institute for Clinical and Health Research (MICHR), and Michigan Medicine through the Health Information Technology and Services (HITS) Department. We are making EMERSE available at no cost for other sites to use.

Before starting

To get EMERSE up and running there are several things you should know.

While the system was designed to be simple to use for non-technical people, installing the system requires significant technical expertise, and will likely need some degree of local customization. We expect those installing EMERSE will have expertise or experience in databases (SQL), Apache Solr, and Java servlet containers such as Apache Tomcat. A strong understanding of security and regulatory requirements is also required.

We have a fully operational system on a virtual machine (VM) that includes PubMed abstracts as an example data source (see http://project-emerse.org for details, including a detailed VM manual). This is good for understanding how the system works, but is not ideal for operational use. We can also provide the system packed up as a .war file which can be placed in to the Tomcat `webapps` directory as described in these instructions. If you want the original source code, we can send an export of an Eclipse project. The source build is easiest if you use the brand of Eclipse that we use (Spring Tool Suite), but it can also be built via Maven/command line.

It is also important to remember that if your institution will be using EMERSE, it is necessary to ensure that all security and privacy regulations are followed. EMERSE should be installed by IT and informatics professionals who are familiar with the requirements for securing protected health information and maintaining the right infrastructure/IT environment to ensure that the data remain secure behind a firewall.

Hardware Requirements

Like most systems, EMERSE will always perform better with faster hardware. However, it should be possible to get a reasonably performing system without major investment. At the University of Michigan our production system is currently running on an 8-core Intel box with 12 GB RAM. Indices are store on a hard disk-based SAN, currently with 3 TB allocated. Solid state drive (SSD) storage would be expensive but would definitely improve the performance of the system. Our informal testing showed that overall performance increased by about 400% when using SSD storage compared to disk.

With local hosting, we estimate an upfront hardware cost of $10-20k, but that range could be very different depending on the local cost of hardware at your institution. Some of this will depend on how many users (especially number of concurrent users) you expect. You could, of course use less expensive hardware, but please remember that the user experience is vital for the success of EMERSE, and fast response times are vital to a good user experience.

We recommend that the Oracle database that supports EMERSE be on a separate server from EMERSE itself, but they could both reside on the same server if needed.

It also possible to run everything on virtual machines but we generally do not recommend that approach. Depending on the VM configuration and host, performance is generally better when the server process connected to the disk is not through a virtualized storage layer. This is because the underlying Solr indexes used by EMERSE should be as low latency/high throughput as is practical.

High Level Overview of Core Components

There are three inter-related but distinct components needed for a full installation of EMERSE: (1) Apache Solr, (2) Document sources (e.g., a document repository), and (3) the EMERSE application itself. It is important to note that components (1) and (2) are necessary for EMERSE to run, but getting these two components up and running are independent of setting up EMERSE. We provide guidance on the issues related to these two components. The three core components are described below in additional detail.

(1) *Apache Solr/Lucene*

EMERSE leverages the Apache Solr project to enable searching of documents. EMERSE requires that any documents go through an indexing process using Solr. Indexing documents involves pushing text and associated metadata to URL's hosted by Solr. Solr in turn generates files in its own unique format (inverted index) that enables fast searching and retrieval. The files are then used by EMERSE, either directly with the Java based Lucene API, or through the REST based Solr API. Note that the EMERSE application itself does not provide functionality to build the index files needed by the application; this is all handled by Solr.

Documents can be pushed to Solr in a number of ways. The Solr project provides a native Java API, a REST API that can be invoked via ETL tools, curl, or other simple HTTP utilities. At Michigan, we primarily use the Java API to index documents, but have also successfully used other ETL tools such as Pentaho Data Integrator (PDI). PDI is currently used to load/index data on our virtual machine (VM) distribution of EMERSE (see http://project-emerse.org/virtualmachine.html).

The Solr project umbrella includes REST based web API's for querying, monitoring and admin tools, as well as the API's for indexing and removing/deleting documents when necessary. The core technology underpinning most of the API's in the Solr project is Apache Lucene, a set of Java libraries. EMERSE uses a mix of the Solr web based API and Lucene libraries.

Additional details about indexing with Solr can be found elsewhere in this manual, see "Indexing with Solr"

(2) *Source Document repository or repositories*

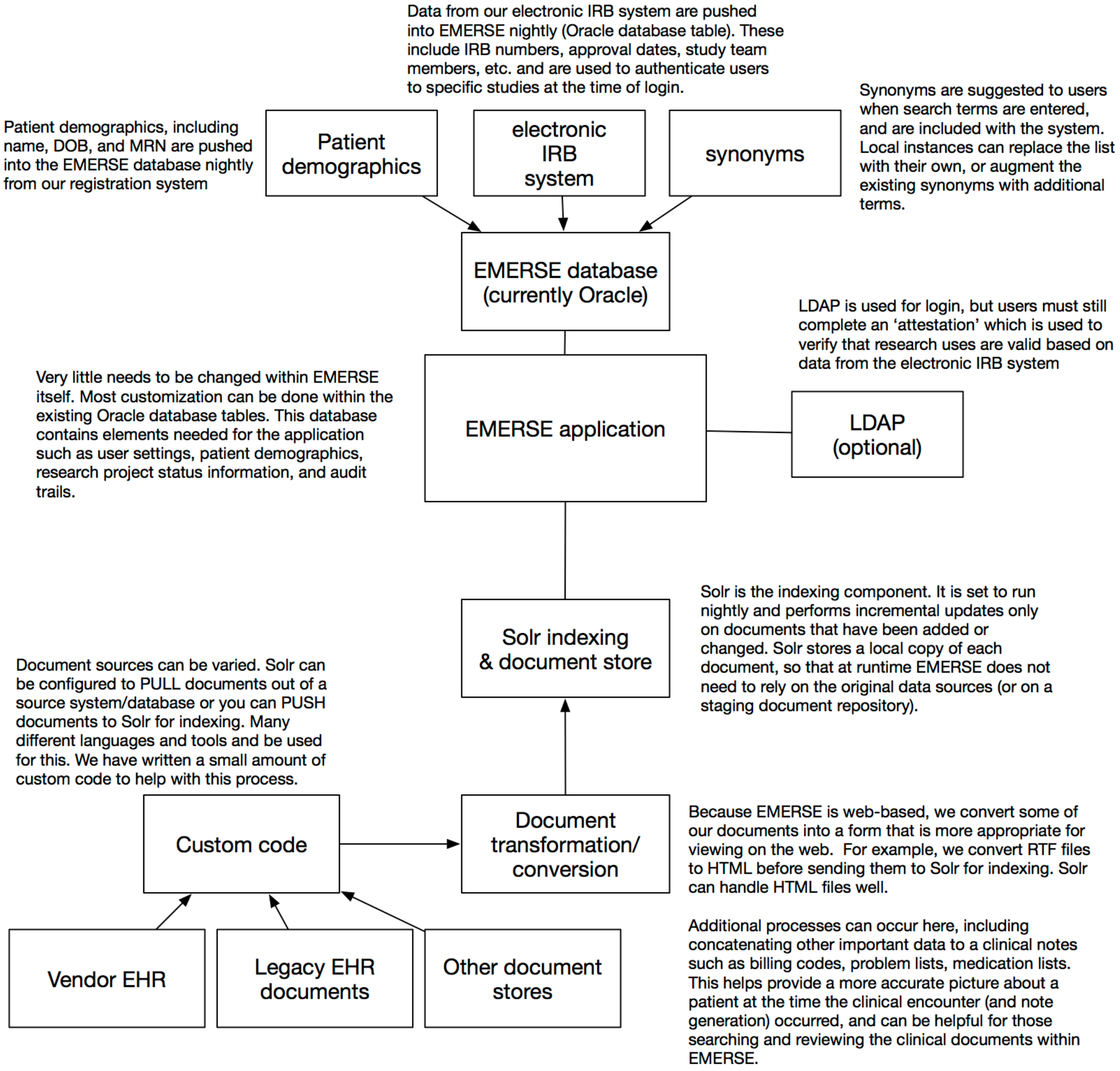
EMERSE is capable of incorporating data from one or more sources in your environment, which likely differs from one organization to another. Existing document stores need to be identified such that they can be accessed and indexed with Solr/Lucene. We recommend that documents are extracted from their original sources, transformed if necessary (for example, conversion from RTF to HTML format), and then stored in a document repository. This document repository would then be used to send documents to Solr for indexing. Use of such a repository should also make it easier to handle and track instances when documents have been changed (e.g., deleted or updated) and is also a more practical approach for times when re-indexing the entire repository is needed. The EMERSE application itself does not need access to these document repositories since, at runtime, EMERSE uses Solr exclusively to search, highlight and present documents and the inverted index contains a copy of the document for display.

(3) *EMERSE*

The EMERSE software runs independently of, but is dependent upon, a functional Solr server with indexes. At startup, EMERSE is pointed to the pre-existing Solr indexes and the Solr server, and then the magic happens. The document content that EMERSE displays to users at runtime comes from the Solr indexes, not the document repository. These indexes need to be accessible to the process (the Java virtual machine) that is running EMERSE but does not need to be on the same server as EMERSE itself. In the case of EMERSE at the University of Michigan the Solr indexes are located on a SAN that is attached to the server. Of course, local storage could also be used.

EMERSE itself does contain an Oracle database separate from the Document Repository. In addition to the general application data about users, audit logs, etc, this database also stores information about research studies and demographic data for the patients in the system. These data come from external sources (an electronic IRB system for the studies, and the electronic health record for the patient demographics), and are pushed into the database each night.

Two additional figures showing high-levels overviews of the setup follow.



Data from our electronic IRB system are pushed into EMERSE nightly (Oracle database table). These include IRB numbers, approval dates, study team members, etc. and are used to authenticate users to specific studies at the time of login.

Patient demographics, including name, DOB, and MRN are pushed into the EMERSE database nightly from our registration system

Synonyms are suggested to users when search terms are entered, and are included with the system. Local instances can replace the list with their own, or augment the existing synonyms with additional terms.

LDAP is used for login, but users must still complete an ‘attestation’ which is used to verify that research uses are valid based on data from the electronic IRB system

Very little needs to be changed within EMERSE itself. Most customization can be done within the existing Oracle database tables. This database contains elements needed for the application such as user settings, patient demographics, research project status information, and audit trails.

Solr is the indexing component. It is set to run nightly and performs incremental updates only on documents that have been added or changed. Solr stores a local copy of each document, so that at runtime EMERSE does not need to rely on the original data sources (or on a staging document repository).

Document sources can be varied. Solr can be configured to PULL documents out of a source system/database or you can PUSH documents to Solr for indexing. Many different languages and tools and be used for this. We have written a small amount of custom code to help with this process.

Because EMERSE is web-based, we convert some of our documents into a form that is more appropriate for viewing on the web. For example, we convert RTF files to HTML before sending them to Solr for indexing. Solr can handle HTML files well.

Additional processes can occur here, including concatenating other important data to a clinical notes such as billing codes, problem lists, medication lists. This helps provide a more accurate picture about a patient at the time the clinical encounter (and note generation) occurred, and can be helpful for those searching and reviewing the clinical documents within EMERSE.

Figure. High-level conceptual overview of the components and data needed to populate and run EMERSE at the University of Michigan. Not all elements are needed (such as the feed from the electronic IRB system).

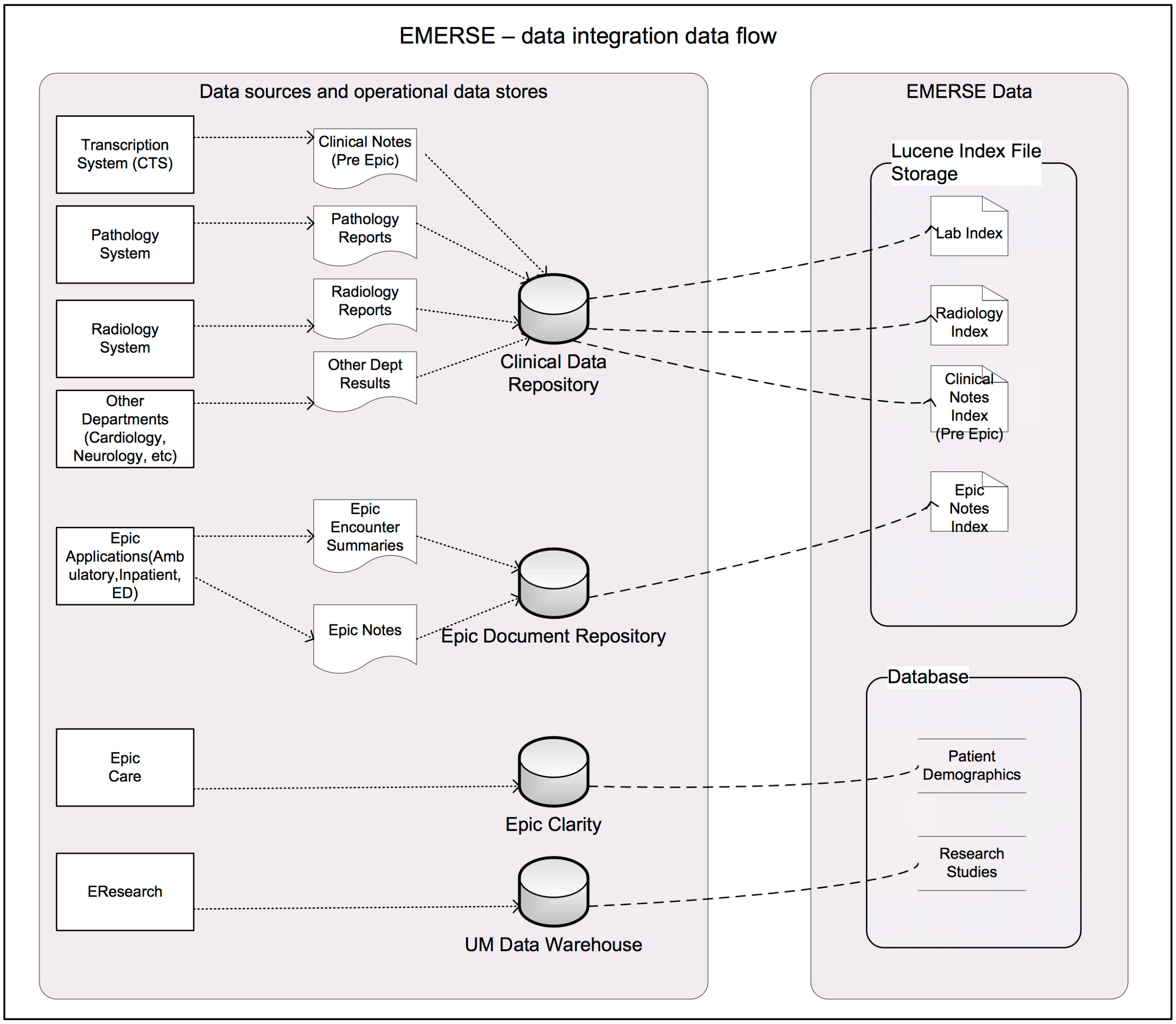


Figure. High-level overview of how data are integrated from various source systems into EMERSE at the University of Michigan. Each institution will likely have its own unique set of sources and integration issues.

Code Dependencies

EMERSE leverages multiple open source components, which are listed below.

Server side dependencies

groupId	artifactId	license	more info
org.springframework	spring-core	Apache 2	
org.springframework	spring-context	Apache 2	
org.springframework	spring-beans	Apache 2	
org.springframework	spring-aop	Apache 2	
org.springframework	spring-tx	Apache 2	
org.springframework	spring-webmvc	Apache 2	
org.springframework	spring-aspects	Apache 2	
org.springframework.security	spring-security-core	Apache 2	
org.springframework.security	spring-security-config	Apache 2	
org.springframework.security	spring-security-web	Apache 2	
org.springframework.security	spring-security-ldap	Apache 2	
org.springframework	spring-jms	Apache 2	
org.aspectj	aspectjrt	Eclipse 1.0	
org.aspectj	aspectjweaver	Eclipse 1.0	
org.springframework	spring-orm	Apache 2	
org.hibernate	hibernate-core	LGPL 2.1	
org.hibernate	hibernate-entitymanager	LGPL 2.1	
org.hibernate	hibernate-envers	LGPL 2.1	
oracle	oracle-driver*	OTN	http://www.oracle.com/technetwork/licenses/distribution-license-152002.html
c3p0	c3p0	LGPL 2.0	
junit	junit	Eclipse 1.0	
org.codehaus.jackson	jackson-core-asl	Apache 2	
org.codehaus.jackson	jackson-mapper-asl	Apache 2	
org.springframework.webflow	spring-js	Apache 2	
org.slf4j	slf4j-log4j12	MIT	
commons-logging	commons-logging	Apache 2	
log4j	log4j	Apache 2	
commons-httpclient	commons-httpclient	Apache 2	
commons-fileupload	commons-fileupload	Apache 2	
commons-io	commons-io	Apache 2	
org.apache.activemq	activemq-client	Apache 2	
org.apache.activemq	activemq-broker	Apache 2	
net.sf.trove4j	trove4j	BSD	
org.springframework.batch	spring-batch-core	Apache 2	
org.springframework.integration	spring-integration-core	Apache 2	
org.springframework.integration	spring-integration-jms	Apache 2	
org.springframework.integration	spring-integration-stream	Apache 2	
org.springframework.integration	spring-integration-jmx	Apache 2	
org.apache.solr	solr-core	Apache 2	
joda-time	joda-time	Apache 2	

Client JavaScript dependencies

name	license	more info
jquery	MIT/GPL	
keyboard.js		https://github.com/RobertWHurst/KeyboardJS/blob/master/license.txt
date.js	MIT	http://www.datejs.com/
json2.js	PUBLIC	http://www.JSON.org/json2.js
amplify.js	MIT	
knockout-2.1.0	MIT	
knockout-validation	MIT	
jquery-ui	MIT/GPL	
jquery-idletimer	MIT	
jquery-json	MIT	
jquery.iframe	MIT	
jquery.fileupload	MIT/GPL	
jquery.balloon	MIT/GPL	
jquery.metadata	MIT/GPL	
jquery.rating	MIT/GPL	

EMERSE Document Repository and Solr Indexing

Document Repository

While not necessary, we recommend a document repository as a staging area for sending documents to Solr for indexing. Documents do not need to be in a single repository, since it is possible to use multiple document repositories if needed. We recommend using a standard SQL database (Oracle, MySQL, PostgresSQL, etc) for the document repository because of its support for backup/restore, etc. Also, we have found that it is easier to use such a database to keep track of document changes that occur after initial indexing (a document might be updated, for example). This allows us to more quickly identify documents that may need re-indexing. In theory it might be possible to point the Solr indexing processes directly to an electronic health record system if the right APIs are in place, but this would likely make it much harder to identify incremental changes in the data and thus might require much broader scanning of the entire source repository each time for any potential changes.

It is also easier to manage data if a document repository is set up. When a new document source is to be added, one only needs to create a new table with the documents and metadata, and then populate the table with the source data. (Of course, one still will need to update Solr/Lucene to point to this new table, and to add information about the source into EMERSE, detailed elsewhere in this document.)

At the University of Michigan we use Oracle for our document repositories (we have more than one), and the repositories are organized by source systems. These repositories contain the documents themselves as well as metadata including *Medical Record Number, Document Date, Last Updated*, as well as details that vary depending on the source such as *Clinical Service, Clinical Provider Name*, etc. These metadata are used in the display for the users within EMERSE, although *Last Updated* is also important for knowing if a document needs to be re-indexed.

Note that this document repository is not a core part of EMERSE and is not something that we set up or include in the EMERSE software. It does not need to be available when EMERSE is running, but the repository must be available when Solr/Lucene is indexing. Solr makes a copy of the indexed documents, so it is Solr that serves up the documents at runtime for EMERSE, not the document repository or source system.

Documents should ideally be in the form of plain text or HTML. At Michigan, since we also receive documents in RTF format, we use a commercial software package (Aspose.Words, http://www.aspose.com/word-component-suite.aspx) to perform this conversion from RTF to HTML for storage in our repository. Other tools exist for other file formats. For example, if PDF documents are stored as source documents, one could use a tool like Apache Tika (https://tika.apache.org) to extract the text to present it to Solr.

To capture clinical notes that reside in Epic, an HL7 based interface that emits HL7 messages containing the note in RTF format when notes are edited and signed has been configured. Details about this process are described elsewhere in this document. A Java process then takes the content of the messages, converts them to HTML using Aspose.Words, and stores it in the document repository. Available metadata in the HL7 message is also stored in the repository along with the note, such as *Encounter Date, Department, and Edit Date*. We recommend that you try to only include finalized, or ‘signed’ notes in the repository to prevent the need to continually monitor for document changes and frequent re-processing and re-indexing. However, such decisions will need to be made locally depending on the local use cases, and needs/expectations of users. At Michigan we do not use Epic’s Clarity repository for our source of documents because Clarity does not preserve the original rich text formatting that makes the documents much easier for users to view within the browser-based EMERSE system.

Getting Documents From Source Systems: Overview

This section provides a general overview regarding the approaches for getting documents from source systems, which is a pre-requisite for submitting them to Solr for indexing. Additional details about specific systems (such as Epic) and Solr indexing can be found in others sections within this document.

Getting documents from source systems will vary considerably at each site and will depend on multiple factors including the number of different sources, how the documents are stored, how they are formatted, the type of access or connections available, etc. It is worth pointing out that documents do not go straight from a source system into the EMERSE application. Rather, documents (whether from a document repository or directly from a source) generally would get pushed to Solr where they are indexed. EMERSE then uses Solr to access the document.

For all documents, the *minimum* elements needed for EMERSE to use them includes:

1. The document text
2. A document date
3. A unique document ID
4. The patient Medical Record Number (MRN) associated with the document

Additional metadata, such as a document type (e.g., “progress note”, “surgical note”) and clinical service (e.g., “general pediatrics”, “rheumatology”), can also be included. These are helpful for users and can be displayed when the documents are listed, and can also be used to sort documents to make them easier to find, and even for filtering results using metadata.

A *Last Updated* date for a document is not needed by EMERSE itself but is very useful to have for setting up the indexing process by Solr so that Solr only updates its index with documents that have been newly added or changed since the last indexing process. We currently handle this incremental updating process through a small amount of custom code that is outside of EMERSE itself. Similar, localized code will be needed at different institutions depending on the source systems and the pathways for moving documents from source systems (or document repositories) to Solr.

Source systems do not have to be live for EMERSE to operate. Documents have to be presented to and indexed by Solr, but Solr/Lucene maintains a local copy of each document so that at runtime EMERSE will use its local copy of the document for display. EMERSE does not need to access the source systems when conducting its searches or displaying the documents to users.

Getting Documents From Source Systems: Epic

Many academic medical centers use the Epic EHR. There are multiple ways in which to get documents out of Epic for indexing within EMERSE, each with its own advantages and disadvantages. Several of these options are described below. Regardless of how the documents are extracted, we recommend storing documents extracted from the EHR in a document repository, and it would be from this interim repository that documents would be moved to Solr for indexing. Such a repository is not needed at run time for EMERSE, since EMERSE stores it's own local copy of the documents (thus there are no real-time calls to Epic when EMERSE is running). Nevertheless, having a document repository will be important if, for example, it becomes necessary to rebuild the indexes, or possibly for comparing existing notes within EMERSE to any new notes that have been generated by the EHR.

Note that regardless of what approach is used for extracting notes, it may be worth concatenating other data to the note to make a larger, more comprehensive, and self-contained note detailing additional aspects of the patient encounter. For example, if a clinical note from an encounters is extracted, it is possible through other queries to extract a medication list at the time of the encounter, append it to the note, and then store this larger, concatenated merged document in a document repository.

1. *HL7 Interface*

The current way in which EMERSE receives encounter summaries and notes from Epic is via an Epic Bridges outbound HL7 interface. Details about this interface can be found here:

http://open.epic.com/Content/specs/OutgoingDocumentationInterfaceTechnicalSpecification.pdf

There may be a fee required by Epic to turn on this interface. At Michigan this interface was turned on to send documents to an external health information exchange (HIE). The emitted HL7 messages include the clinical notes in RTF format, which preserves the formatting and structure of the notes. We have also found these documents to be ideal because, in addition to the actual clinical note, the documents contain other important data relevant to the encounter including the problem list, medication list, billing codes, and other encounter data. This makes searching more comprehensive for users. This is done because multiple "print groups" are included in the outgoing document. These RTF files obtained via this interface are converted to HTML by the commercial software Aspose.Words API, and stored in our document repository for nightly indexing by Solr.

Reference: `http://www.aspose.com/java/word-component.aspx`

One disadvantage of this approach is that it is not useful for loading/extracting historic documents. That is because this outbound document HL7 interface is triggered to send the document based on pre-specified actions (signing a note, for example) and cannot be called on an ad hoc basis.

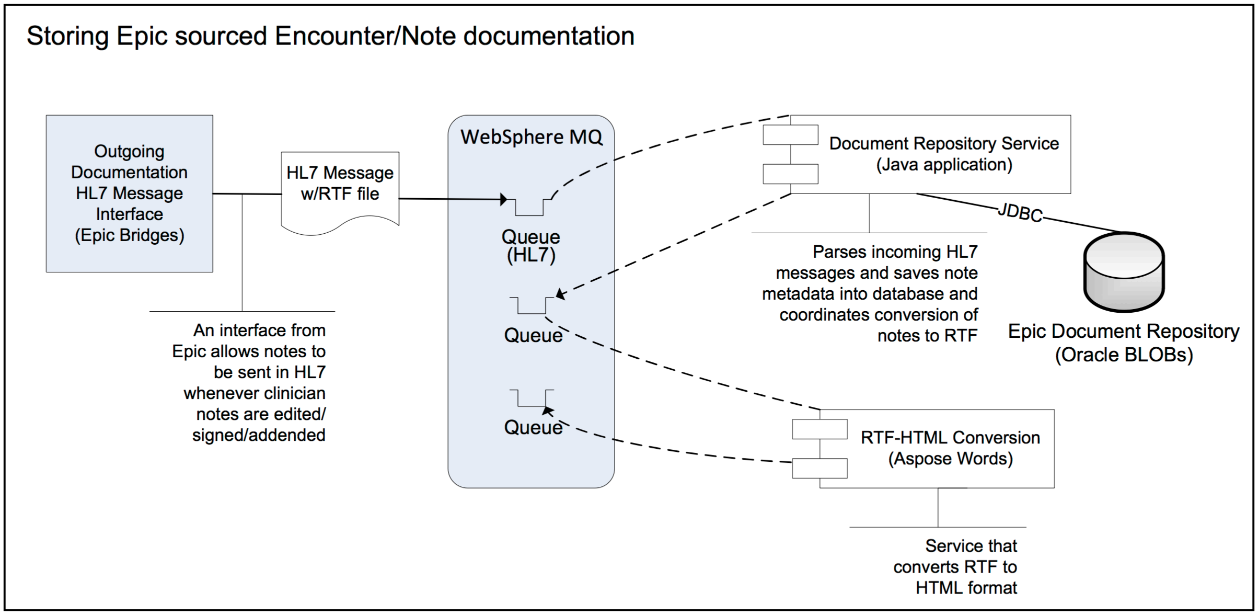

Storing Epic sourced Encounter/Note documentation
Outgoing Documentation HL7 Message Interface (Epic Bridges)
HL7 Message w/RTF file
An interface from Epic allows notes to be sent in HL7 whenever clinician notes are edited/ signed/addended
WebSphere MQ
Queue (HL7)
Queue
Queue
Document Repository Service (Java application)
Parses incoming HL7 messages and saves note metadata into database and coordinates conversion of notes to RTF
JDBC
Epic Document Repository (Oracle BLOBs)
RTF-HTML Conversion (Aspose Words)
Service that converts RTF to HTML format

Figure. The approach used at the University of Michigan to obtain notes from the Epic EHR and make them searchable within EMERSE.

2. *Epic Clarity*

It is also possible to receive plain text notes either through HL7 or through the Epic Clarity database; however, the display of the notes will not look as good as an RTF version since the original document formatting may be essential for understanding/interpreting the document once displayed within EMERSE. Epic Clarity currently does not preserve the formatting, line breaks, etc. but could be considered a source of documents in the future if that were ever to change.

If there is a desire to use Clarity to obtain notes, we suggest looking into the following tables:

`HNO_INFO`	(note metadata)
`HNO_ENC_INFO`	(notes /encounter linkage)
`HNO_NOTE_TEXT`	(actual note text)

These tables don’t seem to be documented in Epic’s Clarity ambulatory care documentation, even though outpatient notes are stored in the same tables along with the inpatient notes as of Epic 2012. However, The inpatient Clarity “training companion” does have an overview of these tables, and they are mentioned in the Clarity data dictionary, so between these two references you should be able to build a query that collects Epic notes.

Note that Clarity might still be useful in order to keep track of what documents have been changed (for example, a note might be deleted or addended weeks after it has been stored and indexed in EMERSE). As far as we are aware, there are no simple ways to call Epic to get a list of what notes have changed within a given time period. Thus, it should be possible to use the Clarity audit tables to detect what notes have changed and then utilize a web service to pull the contents of the new note (see *Epic Web Services*, below). A caveat to this approach, however, is that there is about 1-day delay from the time these changes might have occurred and the time the information gets into Clarity.

3. *Epic Web Services*

It is possible to build a web service to extract documents that preserves the original formatting of the notes. At the University of Michigan we have built a custom web service that extracts formatted notes directly from Chronicles (not Clarity). These notes still have their native formatting which is in RTF, but they are converted to HTML and embedded in a SOAP message during the web service call. However, the note received does not include other useful components such as the problem list and medication list (as described in option 1), although it should be possible to construct a web services that extracts additional “print groups” that could contain other data including the medications and problem list. This approach will require individuals to have experience with writing custom web services and Epic Cache code. Those interested in exploring this option can contact us at the University of Michigan and we can share the Cache code and the web service definition. Note that these web services are not used for EMERSE but are being used elsewhere in our health system. There are likely performance issues with respect to using Web Services for large-scale document retrieval.

Indexing documents with Apache Solr

EMERSE requires that all text documents be indexed in the Lucene format. Indexing can be done in several ways. We use Apache Solr which is a standalone full text server built using Lucene to index documents. Solr provides substantial functionality on top of the Lucene libraries.

Please refer to the installation section of this manual for Solr install instructions ("EMERSE Deployment Guide"). It is assumed that all documents that need to be indexed are available in a data store/document repository with at least the four pieces of metadata required for each document: `MRN, RPT_ID, CLINICAL_DATE`, and `RPT_TEXT` (the clinical document text). While not absolutely required, it is highly recommended that a `LAST_UPDATED` date is also stored so that Solr can perform incremental indexing, since complete re-indexing for many documents is a lengthy process.

There are multiple ways in which documents from source systems can make their way to Solr for indexing, with the primary distinction being either pushed to Solr or pulled by Solr. Both approaches are reasonable depending on local circumstances. Additionally, almost any language can be used to make this happen (Java, Perl, Python, etc).

Three high level approaches are briefly described here, with details in additional sections that follow:

(1) Custom code with SolrJ

> We have found this to be the ideal approach because of it's speed, multi-threaded capabilities, and flexibility.

(2) Solr Data Import Handler (DIH)

> The Solr DIH ships with Solr and should be easy to use. It may be a good approach for getting started and learning Solr. However, while the DIH is useful to understand how the retrieval/indexing process works, the DIH is slow compared to other methods and is not multi-threaded so we do not recommend it for larger-scale implementations.

(3) Non-developer tools

> Documents can also be presented to Solr using non-developer tools such as Pentaho Data Integration tool, which is free and open source, and is used on our demonstration EMERSE virtual machine for loading/indexing data.
>
> http://www.pentaho.com/product/data-integration

Pentaho can read a database and then do a POST to Solr via a REST call. When using the Solr REST API, the JSON/XML/CSV type of data structures are posted via HTTP calls to the Solr server. For our production EMERSE system we use SolrJ, where binary java objects are transmitted, so mapping to these data structures is unnecessary.

Indexing programmatically with SolrJ

SolrJ is a Java client that can be used to add/update a Solr index. This may provide better data indexing speed because multiple threads can be used to load the data. We have found this implementation to be fast and ideal.

Reference: http://wiki.apache.org/solr/Solrj#Adding_Data_to_Solr

A simplified code example of indexing with SolrJ can be found here:

http://www.solrtutorial.com/solrj-tutorial.html

At the University of Michigan we have developed a small amount of custom code to sit between our various source systems and Solr. This custom code is not part of the standard EMERSE release since the specific needs will vary for each institution. The code pulls data from our document repositories via SQL/JDBC and presents it to Solr for indexing via SolrJ. This approach also allows us to do additional transformations as needed such as converting documents in RTF format to HTML format which is ideal for displaying in EMERSE. Another type of possible transformation is concatenating pieces of a document from multiple database rows. This can be done through SQL or through custom code, but it needs to be done before presenting the document to Solr for indexing.

We are using a Unix Cron job to make a REST API call to custom indexing code to start once per night. Because we utilize a last-updated date for each document, the code only indexes new documents that have been added.

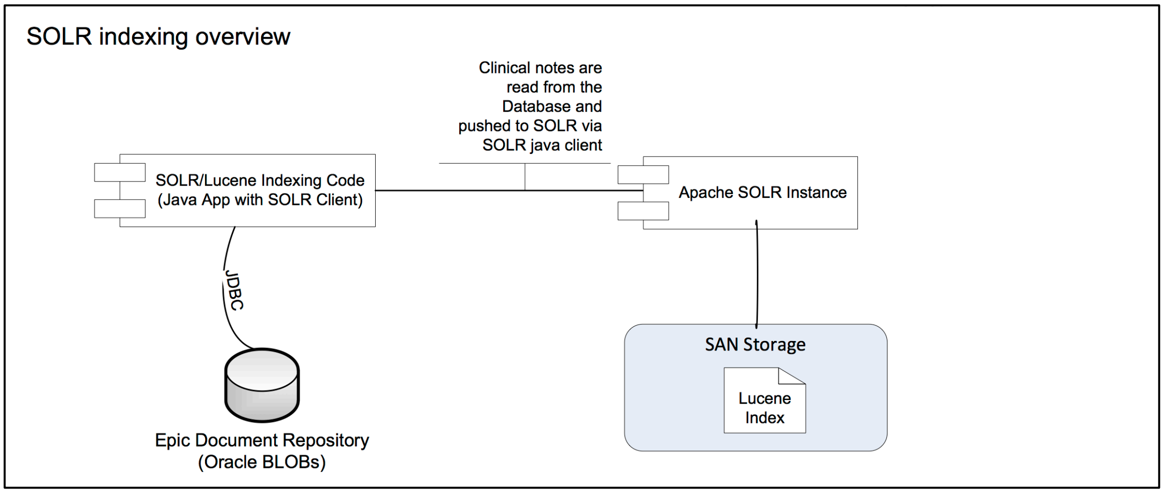

SOLR indexing overview
Clinical notes are read from the Database and pushed to SOLR via SOLR java client
SOLR/Lucene Indexing Code (Java App with SOLR Client)
Apache SOLR Instance
JDBC
SAN Storage
Lucene Index
Epic Document Repository (Oracle BLOBs)

Figure. High-level overview of indexing using Apache SolrJ.

Indexing using Solr DIH (Data Import Handler)

Probably the easiest way to get started with document indexing is to use the Solr Data Import Handler (DIH). This comes packaged with Solr and is easy to get up and running. While the DIH is useful to understand how the indexing process works, the DIH is slow compared to other methods and not multi-threaded so we do not recommend it for larger-scale implementations. The DIH can be configured to access a source system or database, with details about how to retrieve and potentially ‘transform’ the document constructed in SQL. For example, some documents may be in a database with a single document split across multiple rows. The document can be re-assembled into a single text field using SQL and then Solr can process it. Note that Solr expects a document to be ‘pre-assembled’ and does not itself concatenate fields to reconstruct a document from pieces, which is why this should be done in the SQL that retrieves the document. With the DIH you can define the query and the target source in XML, so it will pull documents from that source and present them to Solr for indexing. Solr determines if a document should be added versus updated/replaced based on a unique document key. If you send Solr a document with a previously used key, Solr will replace the older version of the document with the newer version. Thus, when using the DIH in an incremental update scenario, the system would need to be setup to only pull documents that have been updated, which can be done with the SQL ‘select’ statement and a *Last Updated* timestamp.

Reference: https://wiki.apache.org/solr/DataImportHandler

Solr at startup needs a home variable (`solr.home`) defined where it would look for the various configuration files. DIH requires the following two jars found in the `dist` directory of Solr installation:

1. `solr-dataimporthandler-6.x.x.jar`
2. `solr-dataimporthandler-extras-6.x.x.jar`

Database configuration information and the queries needed to retrieve documents are specified in DIH `dataconfig.xml` file. A sample configuration with mandatory fields required by EMERSE is shown below:

```
<dataConfig>
  <dataSource name="XE" driver="oracle.jdbc.driver.OracleDriver" url="jdbc:oracle:thin:@localhost:1521:XE"
   user="system" password="abc" />
  <document name="rpta">
    <entity name="rptsa" pk="RPT_ID" query="select rpt_id,mrn,cast(rpt_date as date) rpt_date,rpt_text,
     cast(clinic_date as date) clinic_date from emerse.reports_ctsa where rpt_type=1"
     transformer="ClobTransformer,DateFormatTransformer">
      <field column="RPT_ID" name="RPT_ID" />
      <field column="RPT_TEXT" name="RPT_TEXT" clob="true"/>
      <field column="MRN" name="MRN" />
      <field column="RPT_DATE" name="RPT_DATE" dateTimeFormat="yyyy-MM-dd HH:mm:ss.S" locale="en" />
      <field column="CLINIC_DATE" name="CLINIC_DATE" dateTimeFormat="yyyy-MM-dd HH:mm:ss.S" locale="en"/>
    </entity>
  </document>
</dataConfig>
```

Additional Solr indexing details

Mapping of the database columns to the Lucene format and how they need to be indexed is specified in `schema.xml`. A snippet of `schema.xml` for the fields specified above in the `dataconfig.xml` above is shown here:

```
<field type="string" name="RPT_ID" indexed="true" stored="true" />
<field type="string" name="MRN" indexed="true" stored="true" />
<field type="date" name="CLINIC_DATE" indexed="true" stored="true" />
<field type="date" name="RPT_DATE" indexed="true" stored="true" />
<field type="text_general_htmlstrip" name="RPT_TEXT" indexed="true" stored="true" termVectors="true"
   termPositions="true" termOffsets="true" />
<field name="RPT_TEXT_NOIC" type="text_general_htmlstrip_nolowercase" indexed="true" stored="true"
   termVectors="true" termPositions="true" termOffsets="true" />
<field name="_version_" type="long" indexed="true" stored="true"/>
<field name="SOURCE" type="string" docValues="true" indexed="true" stored="true"/>
```

A few things to note:

1. There are multiple ways to create indexes depending on which 'analyzer' is used to tokenize the text. Tokenization refers, in part, to the process of how text should be broken up into individual words, and considers properties such as hyphens between words.

2. The Lucene field `RPT_TEXT_NOIC` does not exist in the database query output. The `Copyfield` command of Solr is utilized to make a copy using `RPT_TEXT`. The only difference between these two fields is that text in `RPT_TEXT_NOIC` is tokenized and indexed 'as is' without applying a lowercase filter `<copyField source="RPT_TEXT" dest="RPT_TEXT_NOIC" />`

3. The field type `text_general_htmlstrip` is an extension of Solr `text_general` which uses a HTMLStripCharFilterFactory to get rid of any html tags from the text.

4. The field type `text_general_htmlstrip_nolowercase` is an extension of `text_htmlstrip_nostopwords` where lowercase filterSince we no longer use shards all documents from various sources end up in the same Lucene index. In the unified schema we define a field called SOURCE that points to the original source of documents.

Updating the Solr indexes

We generally do not recommend rebuilding indexes from scratch unless absolutely necessary. For performance reasons we suggest only running incremental updates on the index. We have found that building a new index from many (~100 million) documents can take several weeks. However, nightly incremental updates of new or changed documents (~40,000) generally takes less than 30 minutes. There are a number of ways to detect changed documents in source systems, but the way we do this for EMERSE at Michigan is through the use of audit date fields in the source systems. Because the sources contain a field indicating when it was last updated, this allows nightly batch jobs to only select documents that have changed or been added since the last time it has been run. Generally, the sources also contain a column indicating that a document has been logically deleted. This allows us to remove deleted documents from the associated Lucene/Solr indices.

Note: With incremental indexing, it is necessary to either restart the EMERSE application or reload the indexes for EMERSE (available through the administrator pane) to see the newly added/updated/deleted documents. Here at Michigan, we have set up a REST API call that reloads the indexes every morning via a cron job entry.

Documents sometime also need to be deleted or updated. When a document is deleted through Solr it is actually just flagged as deleted and made unavailable but is not actually deleted from the index. Additionally, there is no true ‘update’ command; however, if an updated document is passed to Solr with the same primary key for a document that already exists, then the prior document will be flagged as deleted and the new document will replace it.

Index Optimization

Over time we have found that many document changes occur as they get updated or deleted (a deletion might be required if, for example, a document was found to be created under the wrong patient). It is possible clear out these deleted/inactivated documents and potentially improve the performance of Solr by Optimizing the documents. This can be invoked manually using the Optimize button in the Solr Administration User Interface. Optimizing also reduces the index segment sizes which can also improve system performance. During the optimization process the original index is left in place while the new, optimized index is being created. This means that you will need empty storage about 2-3 times the original index’s size for optimization to proceed. Additionally, we have found that it can take about 10+ hours to conduct an optimization and it also uses substantial computational resources, meaning that system performance might suffer for users. Thus, it might be best to run this on weekends during times of low use. At Michigan we optimize infrequently and will copy the indexes to a different server with more space and then copy the indexes back after optimization is complete. We also need to ensure that no new documents are added to the original index during this time.

EMERSE Deployment Guide

Deployment Overview

The high level items involved in deploying EMERSE include preparing the application server, initializing the Oracle database, and configuring the Solr/Lucene indices. These instructions will guide you through each of these topics. The diagram below illustrates the high-level architecture of EMERSE. Additional information related to customizing the Solr/Lucene indices to adapt EMERSE to your specific environment can be found later in this document.

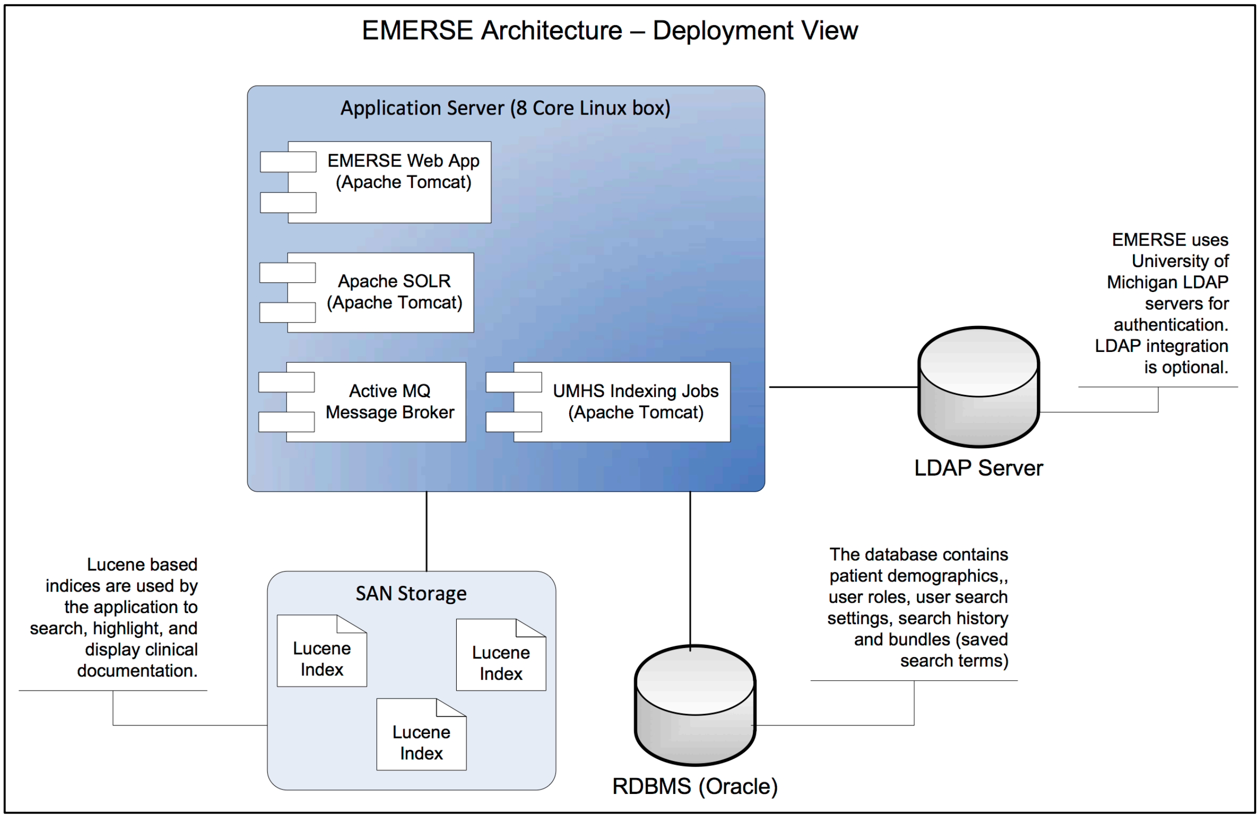

EMERSE Architecture – Deployment View
Application Server (8 Core Linux box)
EMERSE Web App (Apache Tomcat)
Apache SOLR (Apache Tomcat)
Active MQ Message Broker
UMHS Indexing Jobs (Apache Tomcat)
EMERSE uses University of Michigan LDAP servers for authentication. LDAP integration is optional.
LDAP Server
Lucene based indices are used by the application to search, highlight, and display clinical documentation.
SAN Storage
Lucene Index
Lucene Index
Lucene Index
The database contains patient demographics,, user roles, user search settings, search history and bundles (saved search terms)
RDBMS (Oracle)

Figure. High-level architecture diagram.

Pre-Requisites

This guide assumes a few pre-requisite infrastructure items are already in place:

1. An Oracle server should be installed on a server and an account/schema created that allows full access to create common database objects, such as sequences, tables, indices etc. EMERSE doesn't place great demand on the Oracle database, so a relatively small server can be used with 10-50 GB of storage allocated for user tablespaces. Currently at the University of Michigan we are using only 3 GB of space for production EMERSE with 600+ users and 2 years of data.

2. A Linux/Unix based server that will be used to install the application server and host the indexing services. This server should be connected to the highest speed storage available. Capacity is dependent on number of documents to be indexed. At Michigan Medicine 1.5TB is in use to host approximately 100 million documents on the production server. An additional 3 TB is available for index optimization. If your documents are heavily formatted (such as RTF instead of TXT), storage requirements will be higher.

Application Server Installation

This section covers the process of installation and configuring of the application server where the EMERSE application will be deployed. An instance of Apache Active MQ, an EMERSE dependency, will also be installed. The main pieces needing installation are:

1. Java Development Kit/SDK (JDK)
2. Apache Tomcat (Java Servlet Engine)
3. Apache ActiveMQ (A message broker)
4. Solr 6 web application
5. EMERSE Web Archive File (WAR file) deployment and configuration

Specifics for installing each of these components follows.

Installation Instructions: Java JDK

The first step in installing EMERSE is to download and install a Java Development Kit on the server. We recommend version 8 at this time.

Download Site:

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Choose the correct install that matches the target server's operating system. If the target server is a 32 bit OS then choose the 32 bit download option, otherwise choose the appropriate 64 bit install of your system.

Linux:

Download tar binary package based on your Linux system into a desired directory. Untar the package for JDK to be installed using:

```
tar -xvf jdk-xxversion-linux-xnn.tar.gz
```

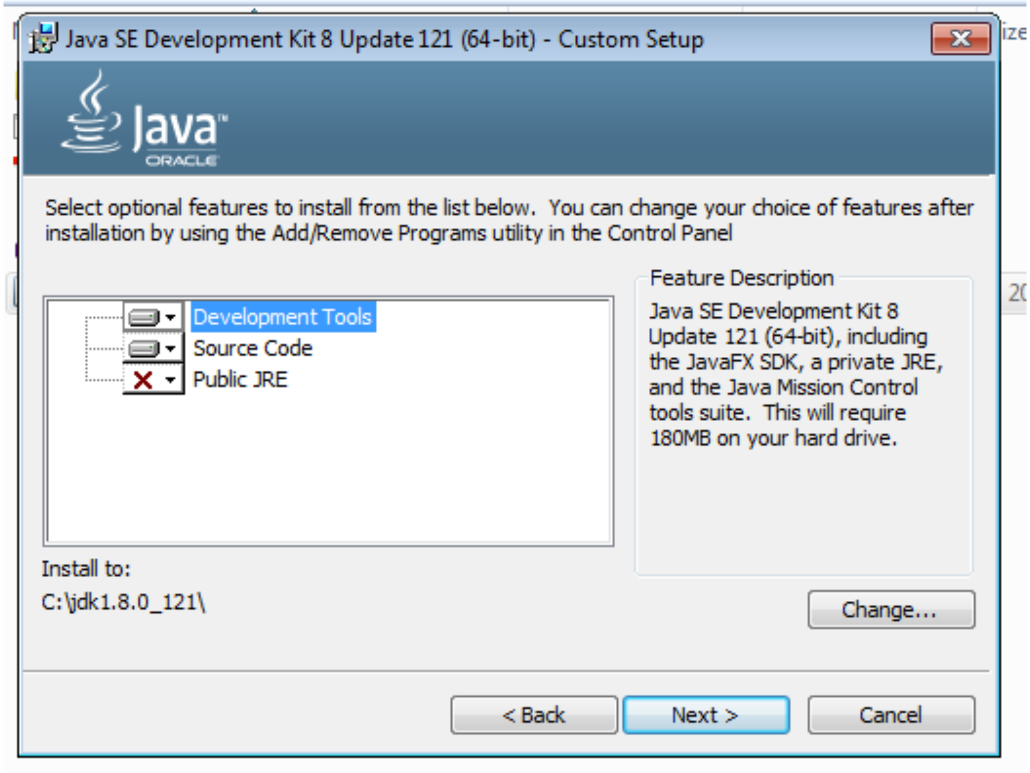
To install the JDK using the RPM binary file, download the file by accepting the license agreement. Install the package using:

```
rpm -ivh jdk-xxversion-linux-xnn.rpm
```

Please note that RPM files need to be installed as root user.

Windows:

After downloading, run the executable. If the target workstation has an existing Java installation that you wish to maintain, unselect the “public JRE” option in the installer as shown below.



Java SE Development Kit 8 Update 121 (64-bit) - Custom Setup

Select optional features to install from the list below. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel

- Development Tools
- Source Code
- ✗ Public JRE

Feature Description

Java SE Development Kit 8 Update 121 (64-bit), including the JavaFX SDK, a private JRE, and the Java Mission Control tools suite. This will require 180MB on your hard drive.

Install to:
C:\jdk1.8.0_121\

Installation Instructions: Apache Tomcat

EMERSE is packaged as a java based WAR file that will be deployed to the Tomcat web server. We have not tested EMERSE with other application servers, but it should run on other servers that support the J2EE servlet specification.

Version: Tomcat 8+

Download site: `https://tomcat.apache.org/download-80.cgi#8.5.9`

Windows:

Download the zip file from binary distributions under the "Core" section. Unzip this file in a desired directory. This will become the Tomcat installed directory. Edit the `startup.bat` file found under `bin` to point to the directory where the JDK was installed.

```
set JAVA_HOME=c:\path_to\jdk_install
```

Linux:

From the binary distributions listed on the page, choose the "Core" tar file. Move the tar file to a desired directory for installation. Extract the tar file using

```
tar zxvf apache-tomcat-8.0.nn.tar.gz
```

Edit the startup script (`startup.sh`) found at `/path/to/tomcat/bin` to point to Java installation directory by adding

```
export JAVA_HOME=/path/to/jdk_install
```

You can also modify JVM settings to use more RAM in the startup script using

```
export JAVA_OPTS="-Xmx2048m -Xms1024m”
```

It has also been observed that EMERSE requires a higher limit on the number of open files than what is typically the default on some Linux systems. This can be set using the `ulimit` command as below prior to starting Tomcat, or added directly to the startup script `startup.sh`.

```
ulimit -v unlimited
```

The exception seen without the higher limit allowance is below:

```
Caused by: java.io.IOException: Map failed
        at sun.nio.ch.FileChannelImpl.map(FileChannelImpl.java:849)
        at org.apache.lucene.store.MMapDirectory.map(MMapDirectory.java:283)
        at org.apache.lucene.store.MMapDirectory$MMapIndexInput.<init>(MMapDirectory.java:228)
        at org.apache.lucene.store.MMapDirectory.openInput(MMapDirectory.java:195)
        at
org.apache.lucene.codecs.compressing.CompressingTermVectorsReader.<init>(CompressingTermVectorsReader.java:118)
        at
org.apache.lucene.codecs.compressing.CompressingTermVectorsFormat.vectorsReader(CompressingTermVectorsFormat.ja
va:85)
        at org.apache.lucene.index.SegmentCoreReaders.<init>(SegmentCoreReaders.java:132)
        at org.apache.lucene.index.SegmentReader.<init>(SegmentReader.java:96)
        at org.apache.lucene.index.StandardDirectoryReader$1.doBody(StandardDirectoryReader.java:63)
        at org.apache.lucene.index.SegmentInfos$FindSegmentsFile.run(SegmentInfos.java:843)
        at org.apache.lucene.index.StandardDirectoryReader.open(StandardDirectoryReader.java:53)
        at org.apache.lucene.index.DirectoryReader.open(DirectoryReader.java:66)
```

Start/Stop:

To start the server use:

```
/path/to/tomcat/bin/startup.sh
```

To stop the server use:

```
/path/to/tomcat/bin/shutdown.sh
```

Installation Instructions: Apache ActiveMQ

EMERSE search requires ActiveMQ for parallel processing of search results from Solr indexes. Whenever EMERSE is running, ActiveMQ needs to be running in the background.

Version: ActiveMQ 5+

Download site: http://activemq.apache.org/download.html

Installation and Configuration:

Download and unzip binary distribution of ActiveMQ. After unpacking, edit the `activemq.bat` (Windows) or `activemq.sh` (Linux) file to point to the directory where the JDK was installed.

Windows:

```
set JAVA_HOME=c:\path_to\jdk_install
```

Start the ActiveMQ broker by opening a terminal/command prompt and navigating to its "`bin`" directory. Type

```
activemq.bat start
```

Linux:

```
export JAVA_HOME=/path/to/jdk_install
```

To start the ActiveMQ broker :

```
cd /path/to/activemq_install/bin
```

```
./activemq start
```

Default port is 8161. Verify it is running by pointing a web browser to:

http://hostname:8161/admin

Installation Instructions: Apache Solr 6

The newest version of EMERSE requires Solr 6 web application to be running so that it can hit the Solr API in real time. This is needed for the ‘All Patient Search’ feature to run.

Version: Solr 6+

Download site: `http://lucene.apache.org/solr/`

Installation and Configuration:

Windows: Download and unzip the zip file to a directory of choice.

To start Solr 6 navigate to the directory where it was installed and type:

```
bin\solr.cmd start
```

Linux: Download the .tgz file and extract it to a directory of choice.

```
tar zxf solr-6.x.x.tgz
```

To start Solr 6:

```
cd /path/to/solr_install/

bin/solr start
```

Default port is 8983. Verify it is running by pointing a web browser to:

http://hostname:8983/solr

More information can be obtained at

`https://cwiki.apache.org/confluence/display/solr/Installing+Solr`

Installation Instructions: EMERSE Web Archive File (WAR file) deployment/configuration

Database Initialization (Oracle)

Provided with the distribution are a set of files, each containing SQL statements that create all needed database objects and sample data that will allow the EMERSE application to startup with a default set of database objects, and sample data in the patients, research studies, synonyms and tables. This scripts should be run as the user and schema setup for the EMERSE application (this will be set by each implementing site), and not a system or sysdba user. These files need to be executed in a SQL query tool in the following order:

1. `create.sql`
2. `auditTables.sql`
3. `sqlToPutBackInModel.sql`
4. `synonymsCreate.sql`
5. `lookupData.sql`
6. `patientData.sql`
7. `indexData.sql`
8. `synonyms_index_subset.sql`
9. `synonyms_subset_50k.sql`

Index setup

Copy supplied files to a directory on the application server. The path to the default directory is

```
/app/indexes
```

(See the next section if you would like to change this default path)

EMERSE Deployment and Configuration

The next step in getting EMERSE up and running after initial installation of the application server and configuration of the database with default settings is to deploy the EMERSE WAR file. To deploy the file, first rename the supplied war file to `emerse.war`, then copy the war file to the `webapps` directory of the Tomcat server. If Tomcat is using default settings, the WAR file will be exploded into a number of files in a directory called `emerse`. This directory includes all the files needed to run the application. You will need to make a change to the settings file to reflect the database that will be used. Inside `WEB-INF/classes` directory of the exploded war file, you will find a file called `project.properties`. This file contains the settings to connect to the database. Update the following values as appropriate for your Oracle database.

For example:

```
ds.username=emerse
ds.password=emersepassword
ds.url=jdbc:oracle:thin:@myhost.med.umich.edu:1521:hostname
ds.driver=oracle.jdbc.driver.OracleDriver
ds.maxPoolSize=10
```

To change path of the indexes directory, update the file `springPostprocessor.properties` found in `WEB-INF/classes` directory.

For example:

```
luceneSearcher.indexPath=/mydir/indexes
```

Once the file is saved, the application server will need to be restarted to reload the configuration to use the latest changes.

Running EMERSE

At this point EMERSE should be up and running. You can verify by pointing a browser to:

`http://hostname:port/emerse`

When the login screen appears provide the following credentials:

Demo user: emerse Password: demouser

Next Steps

Please see the other sections in this technical manual that provides information on how to integrate your institution's data into EMERSE. This manual provides tips on indexing data, and information on loading tables that are unique to your organization.

EMERSE Data Dictionary and System Customization Guide

Background

EMERSE stores its internal system data within an Oracle database. If necessary, it would be possible to change the database to an open source one, although we do not recommend it at this time due to the effort it would take, and this approach is currently untested and unsupported by the EMERSE team. Further, we have worked hard to optimize the way Oracle performs for the specific needs of EMERSE to ensure optimal performance for users.

For the purposes of getting started, Oracle makes available a free “Express Edition” that is fully functional. This free edition of Oracle supports 1 core and up to 10 GB of disk space, which should be enough to support a few users in a demonstration version, or even for a low-powered production version.

http://www.oracle.com/technetwork/database/database-technologies/express-edition/overview/index.html

The primary data stored within this Oracle database includes a patient demographics table, audit logs, and user data including default settings for each user. These are described below.

Note: The large data stores for the documents and document indices are not stored within this database. Instead these are managed by Solr in its own data store and can be on a separate server from the Oracle database.

Patient Demographics

Table: `PATIENT`

Population: From external source (such as EHR)

Population Frequency: Can be variable, but once per day is reasonable

The EMERSE schema includes a patient table with medical record number (MRN), name, date of birth, and other demographic information which is displayed in the search results. Note that the demographics in this table reflect the same as those found within the i2b2 Workbench. Data in this table are used to display the patient name, as well as to validate user-entered MRNs and to calculate current ages of the patients.

At the University of Michigan, this table is updated once per day (overnight) which coincides with the timing of indexing all new/changed notes (also once per day). The Solr indexing process does not require that this table be up to date, but the EMERSE system will need the MRNs to match between the `PATIENT` table and the Solr indexes for everything to work properly when users are on EMERSE.

Note: Currently only MRN, `name`, and `birth_date` are required and used by the system, and thus the other elements are not required. However, near term plans exist to summarize other elements (such as race, gender, etc) to provide more feedback to users about their cohorts.

Column name	Description	Required or Optional
`id`	Primary Key	Required
`external_id`	Medical Record Number	Required
`first_name`	First Name	Required
`middle_name`	Middle Name	Optional
`last_name`	Last Name	Required
`birth_date`	Birth Date -- used to calculate current age	Required
`sex_cd`	Sex	Optional
`language_cd`	Language	Optional
`race_cd`	Race	Optional
`marital_status_cd`	Marital Status	Optional
`religion_cd`	Religion	Optional
`zip_cd`	ZIP code	Optional
`create_date`	Date the row was created. Used to track if the batch job that populates this table ran.	Optional
`update_date`	Date the row was updated. Used to track if the batch job that populates this table ran.	Optional

Research Studies and Attestations

Immediately after each login, every user is required to 'attest' to their use of EMERSE for that session by specifying their reason for using the system. This is called the 'Attestation' page, and the results are stored in the audit logs. EMERSE provides three options (configurable by a system administrator) for this attestation: (1) a free text box, (2) 'Quick Buttons' for choosing pre-selected options that are commonl used (for example, "Quality Improvement", "Patient Care", "Infection Control", etc), and (3) a table of research studies to which a user is assigned.

EMERSE is often used to aid in research studies. While not required, there is a provision for users conducting research to specify their study IRB number at the attestation page after successfully logging in. This enables EMERSE to link that particular session with the study. Loading your institutional IRB data into the `RESEARCH_STUDY` table will enable EMERSE to validate a user's access against the research studies. EMERSE checks to ensure that the study number is valid, that the study's expiration date is not earlier (older) than the current date, and the current study status (since only certain study statuses allow access). Examples of study statuses at Michigan Medicine include "Approved", "Terminated", "Pre Submission", "Expired", "Changes required by core staff", etc. EMERSE also checks that the user is a member of the study team. Note: to leverage this user/study validation feature you may have to customize the list of study statuses to match your local needs (defined in the `VALID_RES_STUDY_STATUS` table, described below).

At the University of Michigan we have been using a commercial IRB tracking system (Click Commerce eResearch). A subset of the data from that IRB system are moved to a data warehouse every night, and we extract a subset of data from that warehouse to bring into EMERSE for validation of users to studies. This is done with custom code that is not a part of EMERSE since it will likely vary at each institution. The code is required to populate the research study tables in EMERSE, shown below. We have generally found that the amount of data in the research system is small enough to completely refresh the EMERSE tables nightly rather than incrementally updating the data.

Also note that with the approach taken at Michigan with nightly updates, there can be a delay in the time between when a study is approved and when a user would be granted access to EMERSE for that study. There is also a small chance that a user could still be permitted to be approved for use on a study even if the study was revoked (again because of the delay introduced by nightly refreshes). This would only happen in cases where a study was terminated or a user was removed from the study before the end date of the study; either way the audit logs would still capture this access.

Ideally there should be no need to change the code within EMERSE to validate if a user should be able to access EMERSE for a research study. Depending on the logic required, this could be determined outside of EMERSE with some custom code and then the result could be used to populate the database with a study status that is considered valid. If, for some reason, it is necessary to modify the EMERSE source code to change way the validation logic is determined, please contact us if assistance is needed.

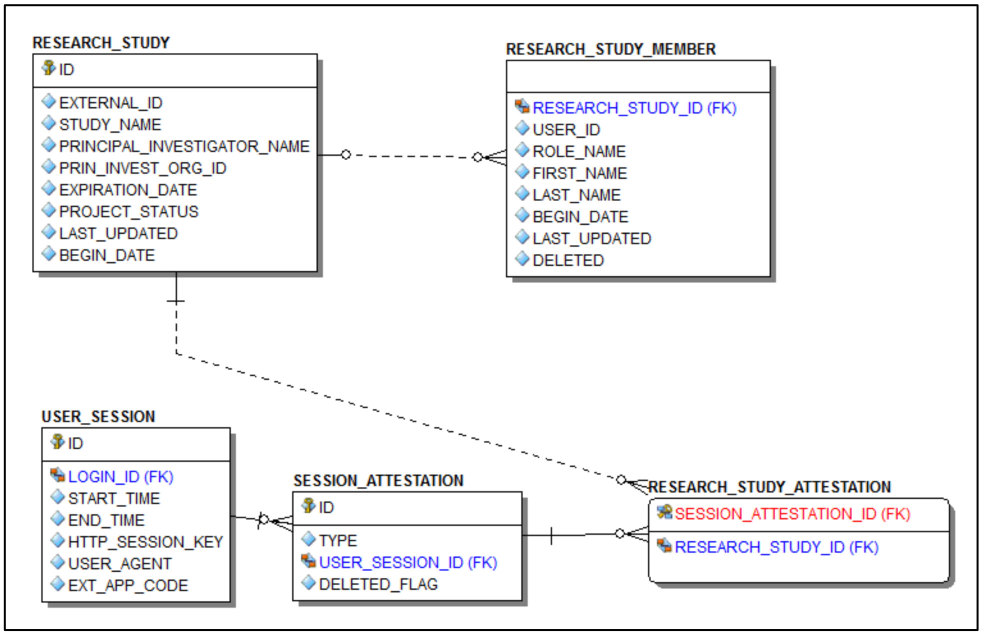


Figure. Entity relationship diagram of some tables related to capturing attestion data which occurs immediately after a user logs in.

Table: `RESEARCH_STUDY`

Population: Populated from external source such as an electronic IRB system

Population Frequency: Can be variable, but once per day is reasonable

This table contains information about studies, and should ideally be populated from an external electronic IRB system. At Michigan this is done nightly, with a complete refresh every night since the amount of data is small enough that incremental updates are not needed.

Column name	**Description**	**Required or Optional**
`id`	Primary Key	Required
`external_id`	IRB study number -- used to link specific studies to usage, and is very helpful for tracking research usage	Required
`study_name`	Name of the study	Required
`principal_investigator_name`	Name of the principal investigator	Required
`prin_invest_org_id`	id of principal investigator. Not currently used by EMERSE.	Optional
`expiration_date`	Expiration date of study. Used to determine if a user should be allowed to proceed. If the expiration date is older than the current date, access should not be granted.	Required
`project_status`	Current project status. This is used by the IRB system to track where a study is in the review and approval process. Only certain study statuses allow access to EMERSE for research. The valid statuses are defined in the `VALID_RES_STUDY_STATUS` table	Required
`last_updated`	A last updated date that comes from the source system. Not currently used within EMERSE.	Optional
`begin_date`	The date the study is allowed to begin. This may also be an approval date. Currently EMERSE does not use this date since being able to start the study is better captured in the status that is obtained from our electronic IRB tracking system. In other words, once a study is approved a user can start their research, until the study reaches its expiration date or the status is change from approved to something else.	Optional

Table: RESEARCH_STUDY_MEMBER

Population: Populated from external source such as an electronic IRB system

Population Frequency: Can be variable, but once per day is reasonable

This table contains information about study team members, and is related to the RESEARCH_STUDY table, described above. Each study can have one or many study team members. This table should be populated in the same manner that the RESEARCH_STUDY table is populated. At Michigan, the data in this table are refreshed nightly, with a full refresh rather than incremental updates.

Column name	Description	Required or Optional
RESEARCH_STUDY_ID	Foreign key reference to row id in RESEARCH_STUDY table	Required
USER_ID	Foreign key reference to row in LOGIN_ACCOUNT table	Required
ROLE_NAME	A string describing a person's role on the study team. EG. "PI", "Staff", "Study Coordinator"	Optional
FIRST_NAME	First name of the username who is on the study. It is currently populated from the source IRB system, but it is not used at all by EMERSE.	Optional
LAST_NAME	Last name of the username who is on the study. It is currently populated from the source IRB system, but it is not used at all by EMERSE.	Optional
BEGIN_DATE	This is not currently used by EMERSE.	Optional
LAST_UPDATED	Date row was last updated	Optional
DELETED	Flag to indicate if the record has been logically deleted	Required

Table: SESSION_ATTESTATION

Population: By EMERSE itself--used internally by EMERSE

Population Frequency: In real time by EMERSE

What is the proper description of this table? (useful to know if someone wanted to query it)

Column name	Description	Required or Optional
id	Primary Key	--
type	A string indicating the top level category of attestation. RSA indicates session is used for research. OTH means other usage. Research attestations will have an associated row in RESEARCH_ATTESTATION. If the type is OTH, a row will also exist in OTHER_ATTESTATION_REASON.	--
User_session_id	A foreign key reference to the user_session table	--

Table: `VALID_RES_STUDY_STATUS`

Population: By System Admin. Only needed if research studies need to be validated.

Population Frequency: May only need to be done once, at the time of system setup. May need periodic updates if the source data (such as from IRB system) defining study status is changed.

EMERSE contains a simple table defining study statues. The statuses that are initially populated in the system (loaded up in the build script) are unique to the University of Michigan (that is, they were developed locally and are implemented in our separate electronic IRB tracking system) and other implementations would have to have their own set of valid statuses if these were to be used to validate and approve usage for research. If the status of a research study is not in this table, EMERSE will not allow the study to be used for attestation; that is, the study would not even be displayed to the user to select.

Column name	**Description**	**Required or Optional**
`status`	A list of study statuses that EMERSE considers valid in terms of allowing a user to proceed. These statuses are generally defined by the IRB and are universal across studies.	Required

`VALID_RES_STUDY_STATUS` Table Example

STATUS
Exempt Approved - Initial
Approved
Not Regulated
Exempt Approved – Transitional

Table: OTHER_ATTESTATION_REASON

Population: ???

Population Frequency: ???

For non-research attestations, there is a lookup table called OTHER_ATTESTATION_REASON that lists available options. These can be configurable by each institution, and may include commonly used access reasons that don't involve research (such as quality improvement, patient care, etc). These options (other than the Free text reason) can be used to populate "quick buttons" that provide a simple way for a user to click on one of the common reasons for use.

Column name	Description	Required or Optional
USER_KEY	?? what is it used for?	Required ??
DESCRIPTION	Foreign key reference to row in LOGIN_ACCOUNT table	Required ??
DELETED_FLAG	Has this reason been deleted? (0=no, 1= yes)	Required ??

OTHER_ATTESTATION_REASON Table Example

USER_KEY	DESCRIPTION	DELETED_FLAG
FRETXT	Free Text Reason	0
RVPREPRES	Review Preparatory to Research	0
STDYDESC	Study involving only decedents (deceased patients)	0

Table: ATTESTATION_OTHER (Not customizable--used internally by EMERSE)

The free text reason that users entered is stored in a table called ATTESTATION_OTHER. This is populated by EMERSE and is not customizable by users.

Column name	Description	Required or Optional
SESSION_ATTESTATION_ID	A unique ID for the session attestation. Used for audit logging.	Required ??
FREE_TEXT_REASON	The free text reason that a user entered.	Required ??
OTHER_ATTEST_REASON_KEY	The attestation reason as it is listed in USER_KEY column of the OTHER_ATTESTATION_REASON table.	Required ??

ATTESTATION_OTHER Table Example

SESSION_ATTESTATION_ID	FREE_TEXT_REASON	OTHER_ATTEST_REASON_KEY
50208	Testing out the system	FRETXT
52060	Testing out the system	FRETXT
46051	Looking up a patient in clinic	FRETXT
71052	infection control monitoring	FRETXT
74107	cancer registry operational work	FRETXT

Clinical Documents

EMERSE search is enabled by the indexing of clinical text documents by Apache Solr. Documents in a clinical environment can come from a myriad of sources like transcription, Radiology, and Pathology, or from an electronic health record. Normally the structure, data, and metadata related to these documents from different sources varies considerably. To simplify things, we configure Solr with a single document schema containing a congolmerate of all fields from all sources. Common data elements, such as patient MRN, clinical date, and source primary key are used across many sources thus are mapped to the same Solr schema field. Search results for each source are displayed in a separate tab in the UI.

Overview:

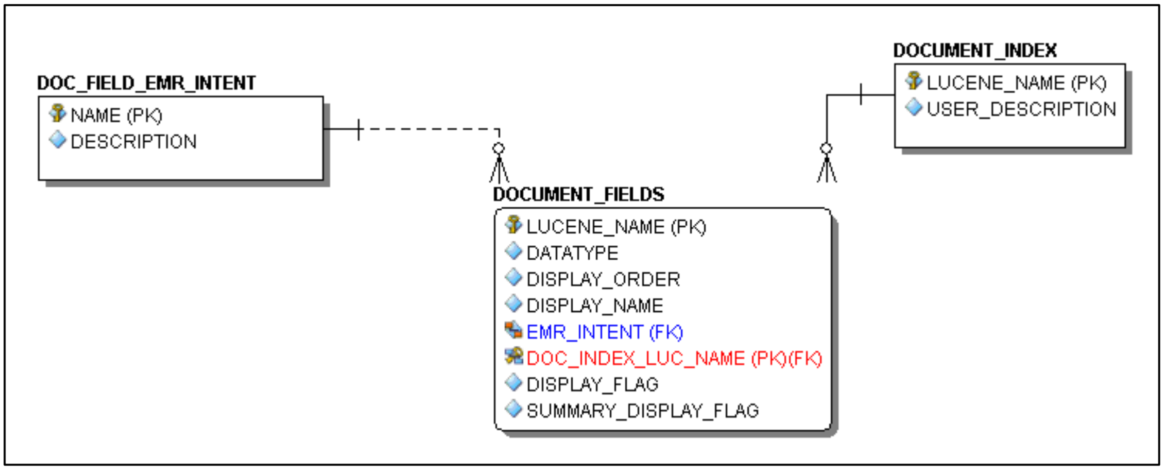


Figure. Entity relationship diagram showing how the three tables above are related.

Table: DOCUMENT_INDEX

Each source of documents (e.g., pathology, radiology, primary EHR, legacy EHR, etc.) is listed as a row in the document_index table. The EMERSE application searches and displays the results based on document source. Document sources normally differ in their format and metadata depending on the source of origin. Each row in this table corresponds to a column in the “Overview” display within EMERSE, and as a subset of documents when a patient is selected.

<table>
<tr><th>Column name</th><th>Description</th><th>Required or Optional</th></tr>
<tr><td>lucene_name</td><td>the name of the document source. This field needs to be unique and search results are displayed on separate tabs for each source.</td><td>Required</td></tr>
<tr><td>user_description</td><td>is the description for the source of document. This field is used when printing individual documents obtained from a search, so that the document source is printed on the document.</td><td>Required</td></tr>
<tr><td>compound_key_flag</td><td>This flag is present for historic reasons and will be deprecated in future releases.
<i>Note:</i>
<i>Lucene indexing requires that each document has a unique identifier. So, when indexing, the fields that uniquely identify a document need to be concatenated and indexed as RPT_ID. For example, when we indexed Radiology documents we used a combination of document Id and exam description to uniquely identify documents. These fields are concatenated using ‘|’ and then indexed as RPT_ID.</i></td><td>Required??</td></tr>
<tr><td>default_sort_column</td><td>is the document field to use to sort the search results for each patient</td><td>Required</td></tr>
<tr><td>display_name</td><td>is the name displayed in the UI</td><td>Required</td></tr>
<tr><td>display_prefix</td><td>is the prefix used by UI components. This can be anything, but each source must have a unique display_prefix.</td><td>Required</td></tr>
<tr><td>display_order</td><td>is the order in which sources appear in the result summary tabs. Each row should have a distinct display order.</td><td>Required</td></tr>
</table>

DOCUMENT_INDEX Table Example

Shown here is a table with sample document_index table data containing three different document sources:

lucene_name	user_description	compound_key_flag	default_sort_column	display_name	display_prefix	display_order
DMI	Central transcription document	0	Case Date	CareWeb	dmi	0
Radiology	Radiology Documents	0	Report Date	Radiology	rad	1
Pathology	Pathology Document	0	Last Updated	Pathology	path	2

Table: `DOCUMENT_FIELDS`

This table provides EMERSE with information about what fields are available in the underlying Solr/Lucene index, their data type, and additional metadata. Each field indexed with Solr/Lucene should exist in this table for each source system in the `document_index` table. The column `EMR_INTENT` is linked to the name field of the `doc_field_emr_intent` mapping table. The column `DOC_INDEX_LUC_NAME` is linked to the `lucene_name` field of the `document_index` table.

Each document source should contain at least six rows (see 'EMR Intent' and the `doc_field_emr_intent` table below for the six required types). One for each type as defined in document fields table. Additional fields can be specified using the generic `EMR_INTENT` options of `TEXT` or `DATE`. These additional/optional metadata fields are used by EMERSE for display in the UI but are not used by Solr/Lucene.

Column name	**Description**	**Required or Optional**
`LUCENE_NAME`	name that corresponds with the Solr document field . The names of the fields are specified in `schema.xml` file	Required
`DATATYPE`	Mainly used by the UI Should be either "Text" or "Date"	Required
`DISPLAY_ORDER`	order in which fields need to appear in the search results	Required
`DISPLAY_NAME`	name that appears in the UI	Required
`EMR_INTENT`	specifies the intent of the field. This refers to the fields defined in the `doc_field_emr_intent` table.	Required
`DOC_INDEX_LUC_NAME`	specifies the document type key from `document_index` table	Required
`DISPLAY_FLAG`	flag that controls if the field is displayed when document is displayed	Required
`SUMMARY_DISPLAY_FLAG`	flag that controls if the field is displayed in search results summary page	Required

`DOCUMENT_FIELDS` Table Example:

Shown below is an example `document_fields` table for three different document sources:

LUCENE_NAME	**DATA TYPE**	**DISPLAY_ORDER**	**DISPLAY_NAME**	**EMR_INTENT**	**DOC_INDEX_LUC_NAME**	**DISPLAY_FLAG**	**SUMMARY_DISPLAY_FLAG**
MRN	Text	0	MRN	MRN	DMI	0	0
RPT_TEXT	Text	1	Report Text	RPT_TEXT	DMI	0	0
RPT_TEXT_NOIC	Text	2	Report Text	RPT_TEXT_NOIC	DMI	0	0
ID	Text	3	Report ID	RPT_ID	DMI	1	1
LAST_UPDATED	Date	4	Last Updated	LAST_UPDATED	DMI	1	0
CASE_DATE	Date	5	Case Date	CLINICAL_DATE	DMI	1	1
MRN	Text	0	MRN	MRN	PATHOLOGY	0	0
RPT_TEXT	Text	1	Report Text	RPT_TEXT	PATHOLOGY	0	0
RPT_TEXT_NOIC	Text	2	Report Text	RPT_TEXT_NOIC	PATHOLOGY	0	0
ID	Text	3	Report Id	RPT_ID	PATHOLOGY	1	1
LAST_UPDATED	Date	4	Last Updated	LAST_UPDATED	PATHOLOGY	1	1
DR_NUM	Text	5	Doctor Num	TEXT	PATHOLOGY	1	1
COLLECTION_DATE	Date	6	Collection Date	CLINICAL_DATE	PATHOLOGY	1	0
MRN	Text	0	MRN	MRN	RADIOLOGY	0	0
RPT_TEXT	Text	1	Report Text	RPT_TEXT	RADIOLOGY	0	0
RPT_TEXT_NOIC	Text	2	Report Text	RPT_TEXT_NOIC	RADIOLOGY	0	0
ID	Text	3	Report ID	RPT_ID	RADIOLOGY	1	1
LAST_UPDATED	Date	4	Last Updated	LAST_UPDATED	RADIOLOGY	1	0
SVC_CD	Text	5	Service Code	TEXT	RADIOLOGY	1	0
DR_NUM	Text	6	Doctor Num	TEXT	RADIOLOGY	1	0
RPT_DATE	Date	7	Report Date	CLINICAL_DATE	RADIOLOGY	1	1

Table: DOC_FIELD_EMR_INTENT (Generally not customizable--used internally by EMERSE)

This is a lookup table for the column EMR_INTENT in the previously defined document_fields table. This table does not normally need to be edited. It is used by the system to help map various sources and types of data to the intended uses of those data by the system. The values contained in the name field of this table are listed below.

Note: The first 6 items are required for the Solr/Lucene indexer to work, the next two are optional, and the final one is no longer used.

Column name	Description	DEFAULT_LUCENE_NAME	Required or Optional
MRN	patient medical record number, which is a unique patient identifier	MRN	Required
RPT_ID	Unique document identifier. This must be unique across all documents and sources	ID	Required
CLINICAL_DATE	Date when the clinical event occurred. Often this would be considered the “note date”	ENCOUNTER_DATE	Required
LAST_UPDATED	Date when the document was last updated, since changes are sometimes made to documents	LAST_UPDATED	Required
RPT_TEXT	The actual text of the clinical document. This field is used by Lucene for lower-case indexing (case-insensitive searching).	RPT_TEXT	Required
RPT_TEXT_NOIC	A copy of the document text to be indexed using a case-sensitive Lucene filter (NOIC = NO Ignore Case)	RPT_TEXT_NOIC	Required
TEXT	Any generic text field. Note that a document may have multiple of these types of generic text fields (e.g., clinical service, document type, clinician name, etc). This is useful when additional metadata are associated with the document and should be displayed.		Optional
DATE	Any generic date field, since a document may have more than one kind of date associated with it.		Optional
~~ENCOUNTER_ID~~	~~This is no longer used. It had been used for a time to search across all patients without limiting it to a set of medical record numbers.~~		~~No longer used~~

Table: LUCENE_SHARDS

Updating: Deprecated--no longer used

Descriptive information is being maintained here until the table is completely removed from the EMERSE system. EMERSE initially used Shards to help with performance, but in no longer does. If this were ever implemented again it would be better to do so using Solr Cloud.

EMERSE used to use this table to locate Solr/Lucene indexes that were available. For most users running EMERSE on a single server, having one row in this table pointing to a single Solr/Lucene index yields adequate performance for 1-2TB indexes with 100's of millions of documents.

Column name	Description	Required or Optional
ID	The Lucene name of the index	Required
PARENT_DOC_INDEX	Specifies the document type key from document_index table. This used to refer to a specific shard.	Optional (Needed when using multiple shards)
START_DATETIME	Start date of clinical documents in this shard	Required
END_DATETIME	End date of clinical documents in this shard	Required

LUCENE_SHARDS Table Example

ID	PARENT_DOC_INDEX	START_DATETIME	END_DATETIME
Unified	(null)	01.02.2008 00:00:00	31.12.2099 00:00:00

Additional Details

Error Logging

EMERSE system errors can be found in the `tomcat_install_dir/logs` directory. They will be in a file called `catalina.out` as well as a file called `emerse.log`. Logging is controlled by a `log4j.properties` file which can be found inside of the expanded `emerse.war` file.

Date Ranges

Our local experience at the University of Michigan has shown that legacy documents coming from older systems may sometimes have invalid document dates. This led to unusual dates being displayed when no date limitation was placed on the search criteria (e.g., “01/01/1900”).

To circumvent this potential problem EMERSE provides two options for controlling the dates displayed to users. In general, background tasks that update the Lucene indexes would also update the date ranges for documents when all dates are selected (that is, when no date range is entered into the date range boxes in the user interface). This is so that as the index updates every night a new ‘end date’ can be shown for the date range of the documents.

This auto-update setting can be over-ridden for the start and stop dates, independently, in the main configuration file, `project.properties`. Changing this setting can, for example, allow one to have a more sensible document start date that more closely matches when the documents were being collected (without having to actually change the dates of all of the incorrect documents).

Note that changing these dates only affects the dates displayed to users. If actual dates are entered by users into the stop/date boxes, those dates will be used. If no dates are entered by users (thus, searching ‘All dates’) then the system will search across all of the documents regardless of the over-ride date shown in the UI and regardless of the document dates in the system.

Security: System Timeouts

The user interface supports system timeouts for periods of inactivity. This is configurable by the system administrators. Timeout is configured in index.html using the JavaScript plugin `idleTimeout`.

An example code snippet is below:

```
$.idleTimeout('#timeoutdialog', 'div.ui-dialog-buttonpane button:first', {
        idleAfter: 3600, // user is considered idle after 60 minutes of no movement
        pollingInterval: 300, // a request to keepalive.php (below) will be sent to the server every 5 minute
        warningLength: 300, // display warning for 5 minutes
        keepAliveURL: './keepalive.html',
        serverResponseEquals: 'OK', // the response from keepalive.php must equal the text "OK"
        onTimeout: function(){
                              }
```

Note: This is a configuration parameter that we would like to move to a more easily modifiable location at some point

Linking the EHR to EMERSE

At the University of Michigan we have developed a link between our vendor EHR (Epic) and EMERSE. A user in Epic can open a patient tab (workspace) and then find the EMERSE button in the ‘More Activities’ menu. Selecting the EMERSE button will automatically open a browser window with the Epic user automatically logged into EMERSE, and with the specific patient loaded into EMERSE and ready for searching.

Security details about how the data and user’s credentials are passed from Epic to EMERSE will vary for each institution. However, the documentation on the Epic site regarding “Integrating External Web Applications into Epic” is an ideal place to start learning about how to do such integrations.

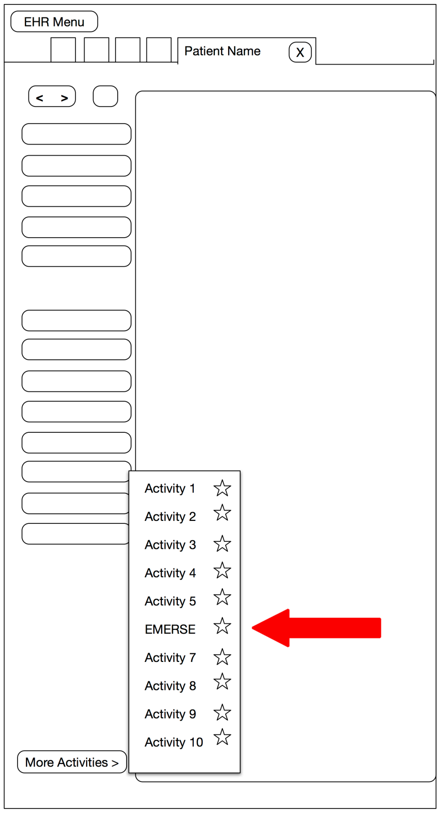
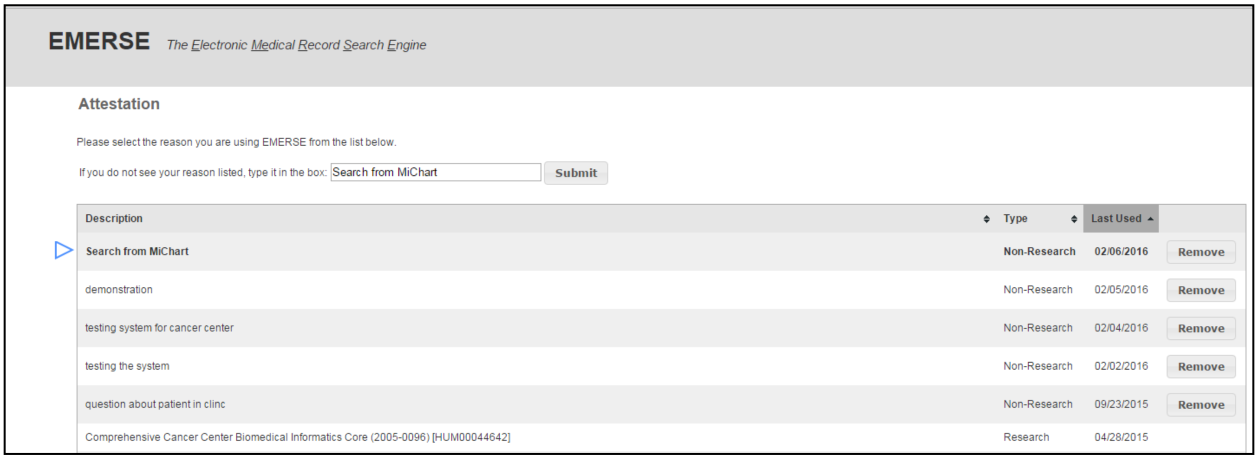


Figure. Partial mocked up/simplified screen showing where the EMERSE activity can be found. Here a patient workspace is open, the More Activities button is selected, and an arrow is pointing to the EMERSE option.



EMERSE *The Electronic Medical Record Search Engine*

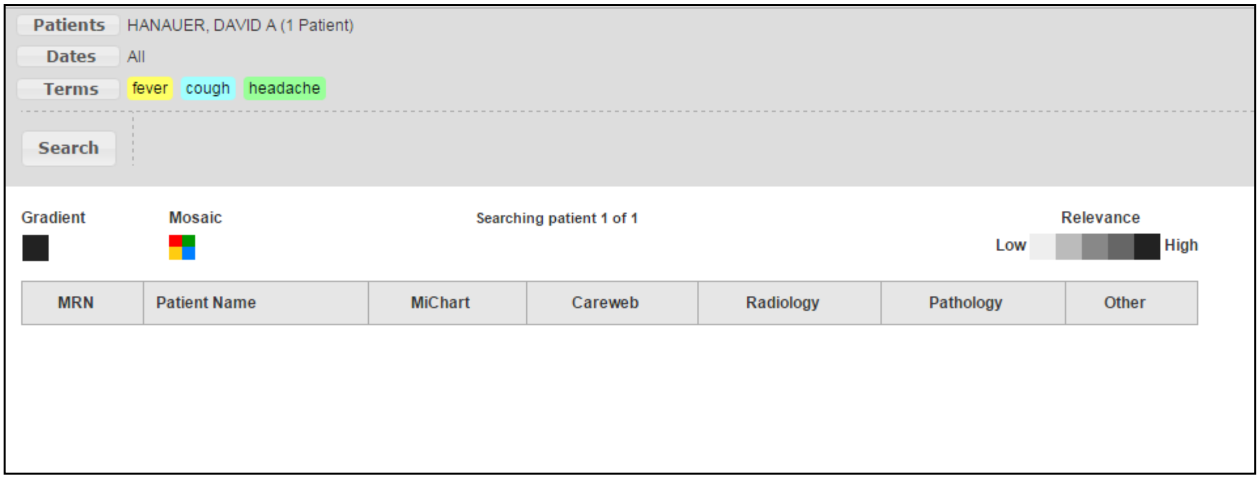
Attestation

Please select the reason you are using EMERSE from the list below.

If you do not see your reason listed, type it in the box: Search from MiChart **Submit**

Description	Type	Last Used ▲	
Search from MiChart	**Non-Research**	**02/06/2016**	Remove
demonstration	Non-Research	02/05/2016	Remove
testing system for cancer center	Non-Research	02/04/2016	Remove
testing the system	Non-Research	02/02/2016	Remove
question about patient in clinc	Non-Research	09/23/2015	Remove
Comprehensive Cancer Center Biomedical Informatics Core (2005-0096) [HUM00044642]	Research	04/28/2015	

Figure. This screen shot of EMERSE shows the first screen a user will encounter when accessing EMERSE from Epic. This is the page where users can enter their reason for using EMERSE for that session, or select from previously entered reasons. In the case of the Epic → EMERSE link, we automatically fill in the option “Search from MiChart” for the user, since our Epic instance is called “MiChart”. However, the user can change it to something more appropriate or specific if desired.



Patients HANAUER, DAVID A (1 Patient)

Dates All

Terms fever cough headache

Searching patient 1 of 1

MRN	Patient Name	MiChart	Careweb	Radiology	Pathology	Other

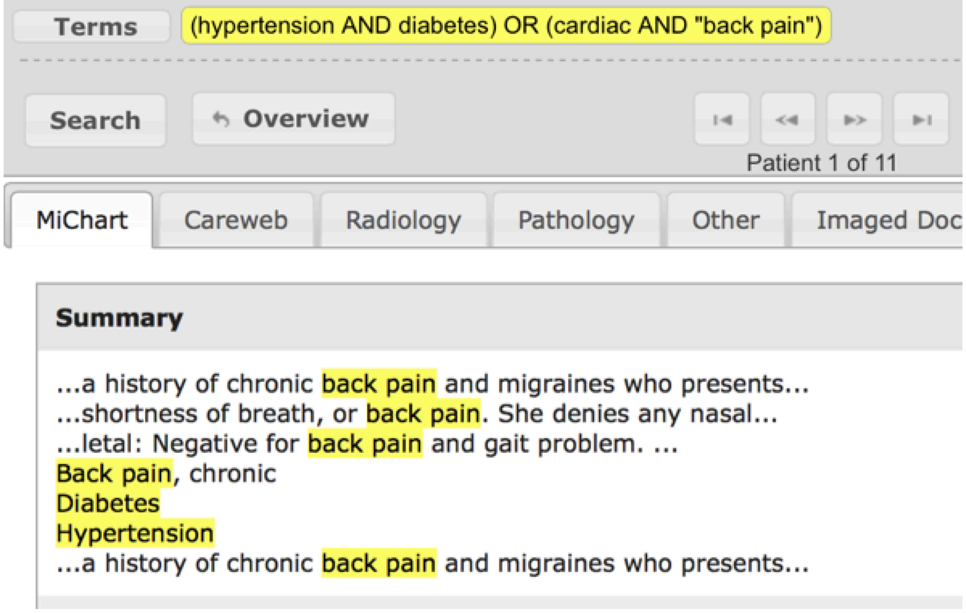
Figure. This screen shot from EMERSE shows the patient already entered in EMERSE, because the medical record number had been passed from Epic to EMERSE.

Known Bugs and Issues

Because EMERSE relies on other open source software, we have identified bugs that we are not able to readily fix. These currently include:

[March 2015]
The 'Advanced Search' feature sends a standard Lucene query without any interpretation or modification by EMERSE. We have found that for complex Boolean queries the system does retrieve the documents correctly, but the separate Highlighter component does not properly highlight based on the Boolean query. Rather it will highlight any terms it finds in the query regardless of the Boolean operation.

The screen shot below demonstrates this. The document was retrieved because 'hypertension' and 'diabetes' were found together in the document. Note that the term 'cardiac' does not appear in the document, but 'back pain' was still highlighted.



(hypertension AND diabetes) OR (cardiac AND "back pain")

Summary

...a history of chronic back pain and migraines who presents...
...shortness of breath, or back pain. She denies any nasal...
...letal: Negative for back pain and gait problem. ...
Back pain, chronic
Diabetes
Hypertension
...a history of chronic back pain and migraines who presents...