EMERSE

Electronic Medical Record Search Engine

Technical Manual

project-emerse.org

David Hanauer <hanauer@umich.edu>

Last updated February 8, 2016

Important Note

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>.

Future plans for EMERSE

For those wondering what is planned for EMERSE, here are a few items:

- Ability to search across all patients, without first defining a patient cohort
- Ability to search imaged/scanned documents
- Filters for reducing results returns (eg, only certain document types)
- Additional synonyms (they are already being added periodically)
- Ability to mark patients as being of interest or not, possibly to record other notes as well.
- New timeline data visualization
- Improved help screens for users
- If adoption beyond U of Michigan occurs, sharing of synonyms and search term bundles across instances
- More security, including the ability to prevent certain users from being able to look up patients (essentially limiting them to specific patient lists)

Still to do 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

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. It was developed at the University of Michigan in 2005 and has been continuously improved and updated since then. EMERSE has been used to support a variety of tasks, including clinical and translational research, internal quality improvement and quality assurance initiatives, as well as for 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 our compliance office, risk management, and infection control departments. It 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, resulting in many peer-reviewed publications.

EMERSE can integrate documents from multiple sources. At the University of Michigan, EMERSE has tens of millions of documents, including those from our original, homegrown EHR system, CareWeb, as well as newer 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 system, EMERSE should be able to search through them. We've even created an interface to move cohorts identified through the i2b2 Workbench directly into EMERSE for further searching of their 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), the University of Michigan Health System through the Medical Center Information Technology (MCIT) Department, and the University of Michigan Medical School through the Medical School Information Services (MSIS) Department. We are making EMERSE available at no cost for academic use, and for a reasonable fee to support ongoing development for other uses.

EMERSE provides features to help get your work done quickly and accurately. For example, the tool includes a large list of synonyms and related keywords to help you expand your search even if you don't know the right terms to use. This includes a large collection of trade and generic drug names, acronyms and abbreviations, and other wording variations including spelling errors.

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

Before you start

To be able 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 Lucene and Solr, and web servers such as Apache Tomcat.

Once a license has been signed (no cost for academic use) we can provide a fully operational system on a virtual machine that includes PubMed abstracts as an example data source. 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 the instruction. 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 to search clinical documents, 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 a 4-core Intel box with 12 GB RAM. Indices are store on a hard disk-based SAN, currently with 2 TB allocated. Solid state drive (SSD) storage would be expensive but would definitely improve the performance of the system. We estimate that performance would be increased by a factor of 2, potentially more.

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, and fewer users. 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 Lucene 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. 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 do provide guidance on the issues related to these two components. It is also worth noting that components (1) and (2) would likely be required for any type of search engine you would implement. A high-level conceptual diagram of the various components is also shown below in the Figure.

(1) *Document repository or repositories*

This is needed for Lucene to locate and index documents. This repository is not needed at run-time for EMERE since it is not accessed by EMERSE. Rather, Lucene will create its own local copy when creating the indices. Lucene can be pointed to multiple sources, and it is not entirely necessary that you create a standalone document repository (you could, for example, connect Lucene directly to your EHR using whatever APIs are available). However, we recommend use of the repository since it will make it easier to handle and track instances when documents have changed (last updated, for example) as well as complete re-indexing (as opposed to incremental indexing) which may be required from time to time.

(2) *Apache Lucene*

EMERSE leverages the Apache Lucene library (jar files) to tokenize document texts in a format that enables fast retrieval. When users execute a search within EMERSE, the application code uses the Lucene API to open the index files and search for matching documents. The EMERSE application itself does not provide functionality to build the Lucene index files needed by the application. At UMHS, Apache SOLR is used to aid with the creation and maintenance of Lucene formatted indexes. It is a web based application that provides REST oriented API's that enable adding and removing content from Lucene indexes. At UMHS, as data from source systems changes, the changed documents are retrieved from the system and sent to the SOLR API for inclusion in EMERSE.

(3) *EMERSE*

The EMERSE software runs independently of, but is dependent upon, functional Lucene indices. At startup, EMERSE is pointed at the pre-existing Lucene indices, and then the magic happens. The text document content that EMERSE displays to users at runtime comes from the Lucene indexes, since Lucene stores a local copy of all documents it indexes. 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 Lucene files are located on a SAN that is attached to the server. Of course, local storage could also be used.

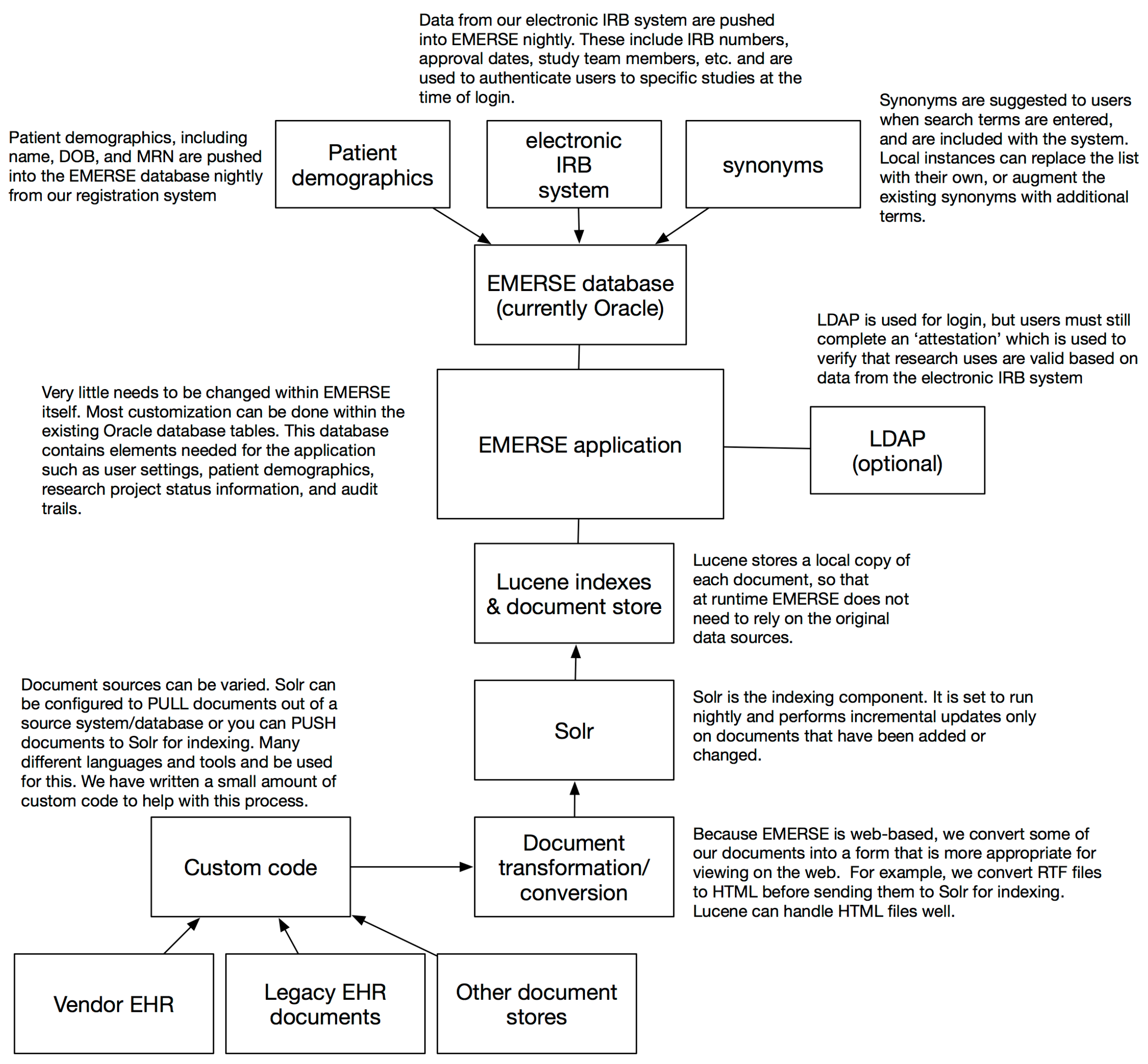

Data from our electronic IRB system are pushed into EMERSE nightly. These include IRB numbers, approval dates, study team members, etc. and are used to authenticate users to specific studies at the time of login.
Patient demographics, including name, DOB, and MRN are pushed into the EMERSE database nightly from our registration system
Patient demographics
electronic IRB system
synonyms
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.
EMERSE database (currently Oracle)
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.
EMERSE application
LDAP (optional)
Lucene indexes & document store
Lucene stores a local copy of each document, so that at runtime EMERSE does not need to rely on the original data sources.
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.
Solr
Solr is the indexing component. It is set to run nightly and performs incremental updates only on documents that have been added or changed.
Custom code
Document transformation/ conversion
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. Lucene can handle HTML files well.
Vendor EHR
Legacy EHR documents
Other document stores

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 necessarily needed (such as the feed from the electronic IRB system).

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	

Document Repository

While not entirely necessary, we recommend a document repository for Lucene to use. Documents do not need to be in a single repository, since it is possible to point Lucene to multiple repositories if needed. We recommend using a standard SQL database (Oracle, MySQL, PostgresSQL, etc) 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 indexing. This allows us to more quickly identify documents that may need re-indexing. In theory it could be possible to point EMERSE directly at an electronic health record system if the right APIs are in place, but this might make it much harder to identify incremental changes in the data and thus might require much broader scanning of the entire repository each time for any potential changes.

It is also easier to manage data this way. 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 Lucene to point to this new table, and to add the source into EMERSE as well.)

At the University of Michigan we use Oracle for our document stores (we have more than one), and the stores are organized by source systems. These stores include the documents themselves as well as metadata including medical record number, document date, last updated, as well as details that depend 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 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 is, however, a pre-requisite for Lucene to be able to retrieve and index the documents. It actually does not need to be running for EMERSE to function, but must be up and running when indexing by Lucene is occurring. Lucene makes a copy of the document it indexes, so Lucene serves up the documents at runtime for EMERSE, not the document repository.

Documents should ideally be in the form of plain ASCII 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.

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. A java process then takes the content of the messages, and 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.

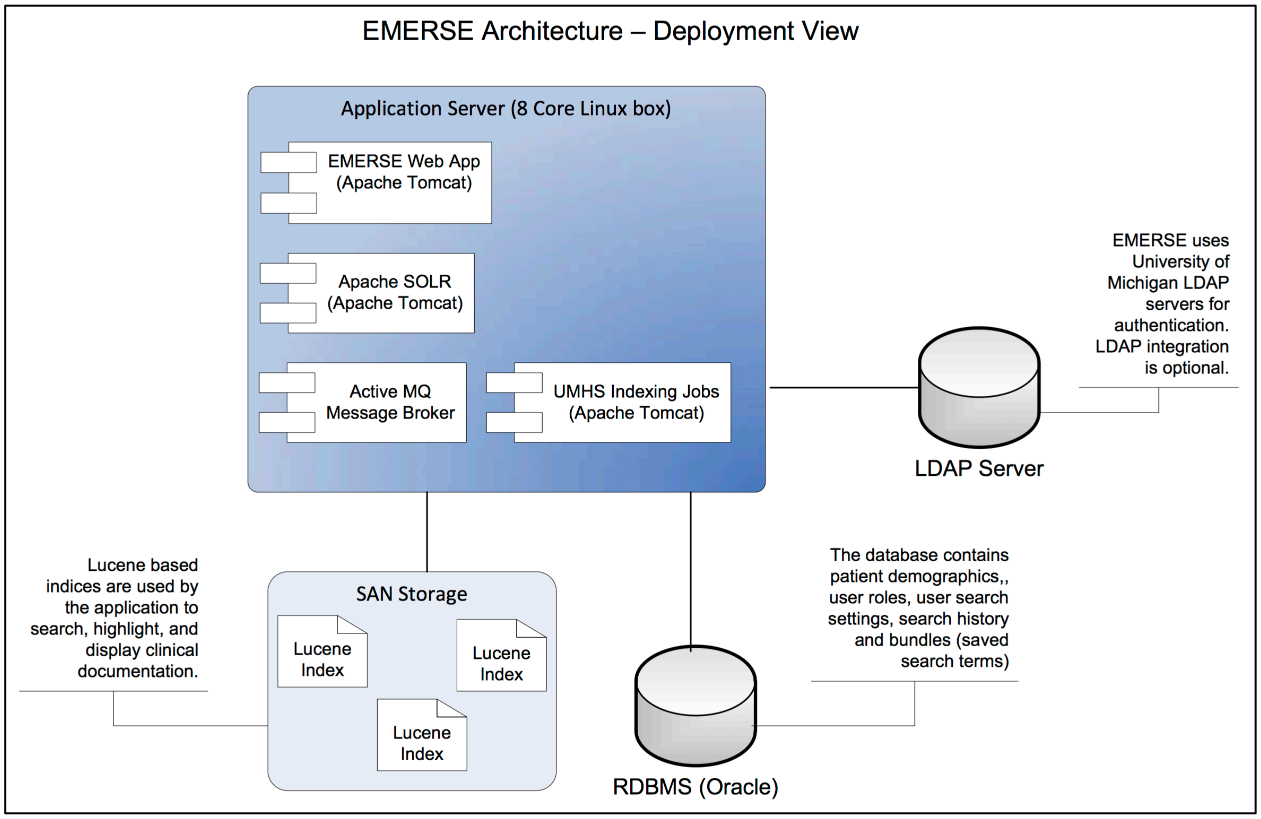
A small amount of Java code was written to retrieve the documents from the repository and other UMHS datasources via JDBC, and execute the SOLR API to update EMERSE's Lucene indexes. This code is executed periodically to keep the Lucene indexes up to date. This code is not part of the standard EMERSE release since the specific needs will vary for each institution.

For example, if PDF documents are stored, one could use a tool like Apache Tika to extract the text to present it to Solr.

EMERSE Deployment Guide

Deployment Overview

The high level items involved in deploying EMERSE include preparing the application server, initializing the database, and configuring the 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 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 gigs 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 UMHS 1 TB is in use to host approximately 100 million documents. 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. EMERSE Web Archive File (WAR file) deployment and configuration

1. 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 computer's operating system. If the target workstation 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
```

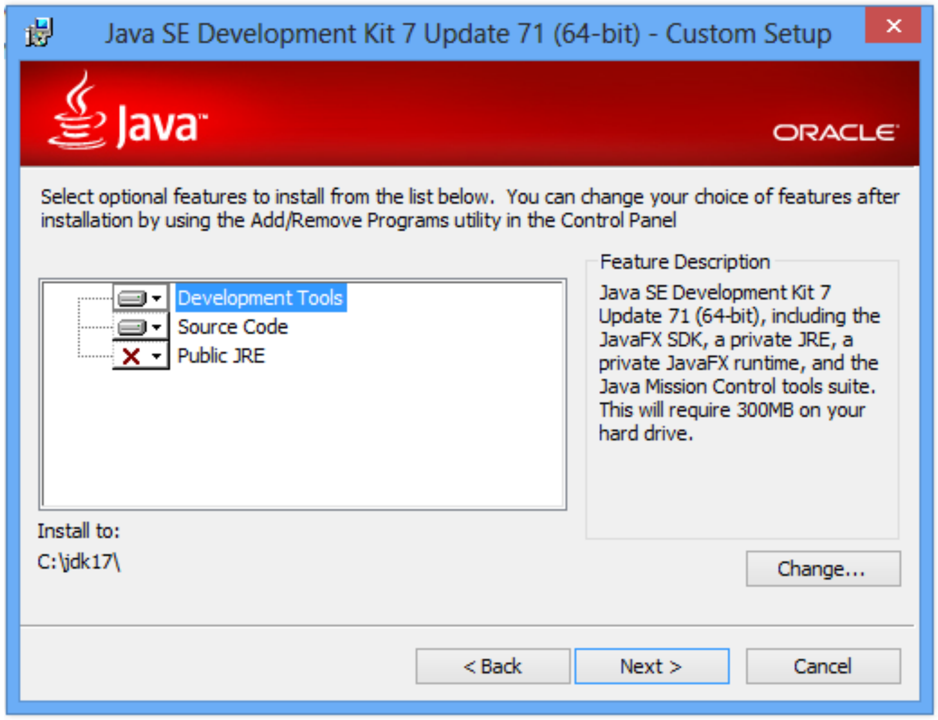
To install the JDK using the RPM binary file, download the file by accepting the license agreement. Install the package using:

```
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.



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

Feature Description

Java SE Development Kit 7 Update 71 (64-bit), including the JavaFX SDK, a private JRE, a private JavaFX runtime, and the Java Mission Control tools suite. This will require 300MB on your hard drive.

Install to:
C:\jdk17\

2. 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 7+

Download site: http://tomcat.apache.org/download-70.cgi

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-7.0.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="-Xmx3072m -Xms1024m”
```

It has also been observed that Solr requires a higher limit on the number of open files to be higher than what is typically the default. This can be set using the ulimit command as below.

```
ulimit -v unlimited
```

The exception seen without the higher limit 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
```

3. Apache ActiveMQ

EMERSE search requires ActiveMQ for parallel processing of search results from Lucene 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

4. EMERSE Web Archive File (WAR file) deployment and configuration

Database Initialization

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. 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. `synonymIndexData.sql`
9. `synonymsData.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. We 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

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 file called `emerse.log`. Logging is controlled by a `log4j.properties` file which can be found inside of the expanded `emerse.war` file.

Next Steps

Please see the accompanying guide that provides information on how to integrate your institution's data into EMERSE. It provides tips on indexing data, and information on loading tables that are unique to your organization.

EMERSE Data and System Customization Guide

EMERSE Data

EMERSE stores its internal system data within an Oracle database. If necessary, it is 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.

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 database includes a patient demographics table, audit logs, and user data including default settings for each user. These are described below. The large data stores for the documents and document indices are not stored within this database. Instead these are managed by Lucene in its own data store.

Patient Demographics Data

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. In addition to displaying the name, the table is used to validate user-entered MRNs and to calculate current ages of the patients. It should be noted that currently only `MRN`, `name`, and `birth_date` are required and used by the system, and thus the other elements are not required.

`PATIENT` Table Details

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

Research Studies Data

EMERSE is often used to aid in research studies. 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). At the University of Michigan we have been using a commercial IRB tracking system, and we extract a subset of data from that system to bring into EMERSE for validation of users to studies. Examples of study statuses at Michigan include "Approved", "Terminated", "Pre Submission", "Expired", "Changes required by core staff", etc. Note: to use this feature you may have to customize the list of study statuses to match your local needs.

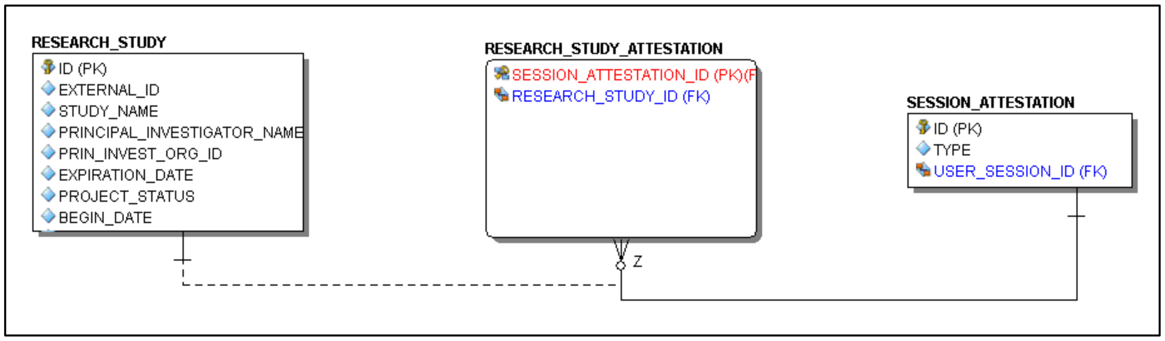


Figure. Entity relationship diagram of some tables related to capturing login attestations.

`RESEARCH_STUDY` Table Details

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
`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
`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.	Optional

`SESSION_ATTESTATION` Table Details

Column name	Description	Required or Optional
`id`	Primary Key	Required
`type`	This is the type of usage as defined by the UI of EMERSE, where a user selects their reason for using EMERSE. For example, research, decedent-only study, other. This is recorded for tracking purposes.	Required

`VALID_RES_STUDY_STATUS` Table Example

The following statuses are currently loaded into the *VALID_RES_STUDY_STATUS* table. These research study valid statuses are loaded up in the build script. These 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.

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

For non-research attestations, there is a lookup table called `OTHER_ATTESTATION_REASON` that lists available options:

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

The free text reason that users entered is stored in a table called `ATTESTATION_OTHER`. An example of what the table would look like with real data is:

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 Lucene. 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. Search results for each source are displayed in a separate tab in the UI.

Documents to Database mapping

Document Index: Each source of documents 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 “heatmap” display, and as a subset of documents when a patient is selected.

`DOCUMENT_INDEX` Table Details

Column name	**Description**	**Required or Optional**
`lucene_name`	the name of the document source. This field needs to be unique and search results are displayed on separate tabs for each source.	Required
`user_description`	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.	Required
`compound_key_flag`	is a boolean flag that is set when a unique document identifier is a combination of more than one field in the document. This is needed if there is no other unique identifier for the documents. 0 = not needed; 1 = needed In other words, if the primary key of a document source consists of more than one field, this flag needs to be set to true. This flag is used internally to set document viewed flag when a user views a particular document. Regarding compound keys, EMERSE just uses the RPT_ID field when searching/fetching the documents from Lucene. The only time this compound flag is used is to track if the document has been viewed by a particular user. The keys are split up and stored in the `document_view` table to easily reference back to the source of documents if need be for auditing purposes. 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`.	Required
`default_sort_column`	is the document field to use to sort the search results for each patient	Required
`display_name`	is the name displayed in the UI	Required
`display_prefix`	is the prefix used by UI components. This can be anything, but each source must have a unique display_prefix.	Required
`display_order`	is the order in which sources appear in the result summary tabs	Required

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

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. This table consists of a column called `EMR_INTENT` that 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.

At the very least, each document source should contain 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 metadata fields are used by EMERSE for display in the UI but are not used by Lucene.

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
RPT_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
RPT_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
RPT_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

A description of the columns names is as follows:

Column name	Description	Required or Optional
`LUCENE_NAME`	name of the document field that was indexed in Lucene. The names of the fields are specified in `schema.xml` file for Lucene to index documents and need to match in this table column	Required
`DATATYPE`	type of the document field	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`	Required

	table.	
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

EMR Intent

The doc_field_emr_intent 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 that the first 6 items are required for the 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	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

_Lucene Shards

EMERSE uses this table to locate SOLR/Lucene indexes that are 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.

`LUCENE_SHARDS` Table Details

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	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

A sample `LUCENE_SHARDS` table is shown below

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

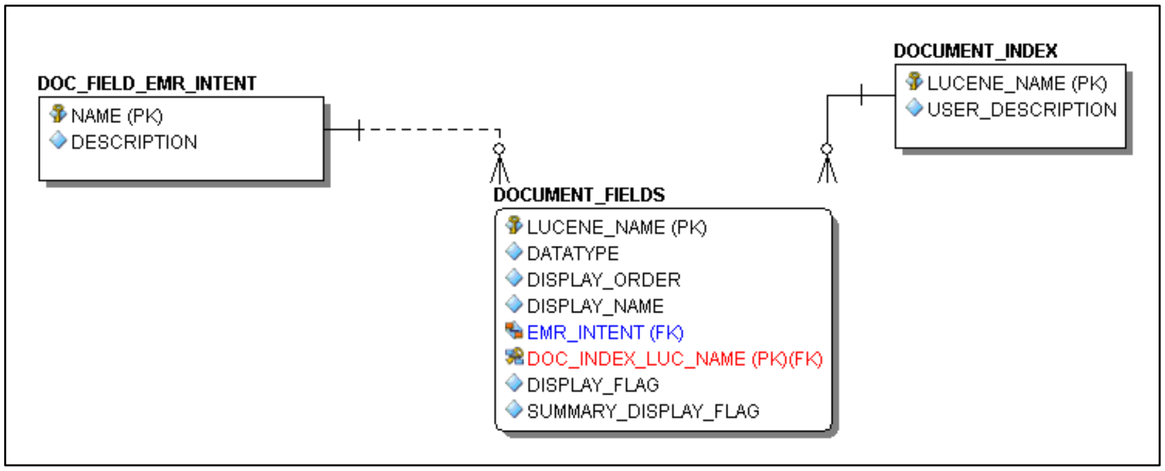


Figure. Entity relationship diagram showing how the three tables above are related.

Getting Documents From Source Systems to EMERSE (via Solr)

One of the most important aspects that any institution will have to figure out is how to get documents from source systems into EMERSE. This 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, etc. 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 Medical Record Number 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. This also provides additional metadata for users to sort documents to make certain ones easier to find for users.

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 only documents that have been newly added or changed since the last indexing process get updated. We currently handle this incremental updating process through a small amount of custom code that it outside of EMERSE itself. Similar, localized code will be needed at different institutions depending on the source systems.

Source systems do not have to be live for EMERSE to operate. Documents have to 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.

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 described:

(1) Probably the easiest way to get started with indexing is to use the Solr Data Import Handler (DIH), described in more detail in a different section of this document. This comes packaged with Solr and is easy to get up and running. While the DIH is useful to understand how the retrieval/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 do that, 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, Solor 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 the SQL ‘select’ statement.

(2) At the University of Michigan we have developed a small amount of custom code to sit between our various source systems and Solr. This code pulls data from source systems and presents it to Solr for indexing via SolrJ, desribed below. SolrJ (J= Java) is multi-threaded and we have found it to be fast. 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.

(3) Documents can also be presented to Solr using non-developer tools such as Pentaho Data Integration tool, which is free and open source:

`http://www.pentaho.com/product/data-integration`

Pentaho can read a database and then do a POST to Solr via a REST call.

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 provide a lot of functionality on top of the Lucene libraries. In the case of indexing Solr provides:

1. A REST-based API
2. Remote Java acces via SolrJ

When using the REST API, the JSON/XML/CSV type of data structures are posted via HTTP calls to the Solr server. We use SolrJ, where binary java objects are transmitted, so mapping to these data structures is uncecessary.

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

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

Please refer to the installation section of this manual for Solr install instructions. It is assumed that all documents that need to be indexed are available in a data store with at least the four pieces of metadata information required for each document: `MRN, RPT_ID`, `CLINICAL_DATE`, and `LAST_UPDATED` along with clinical document text.

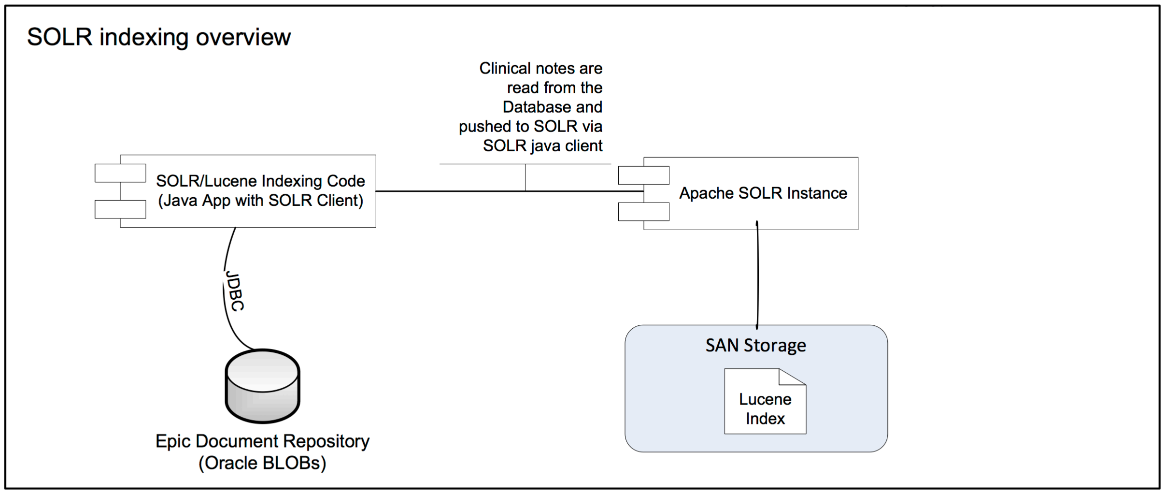

SOLR indexing overview
Clinical notes are read from the Database and pushed to SOLR via SOLR java client
SOLR/Lucene Indexing Code (Java App with SOLR Client)
Apache SOLR Instance
JDBC
SAN Storage
Lucene Index
Epic Document Repository (Oracle BLOBs)

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

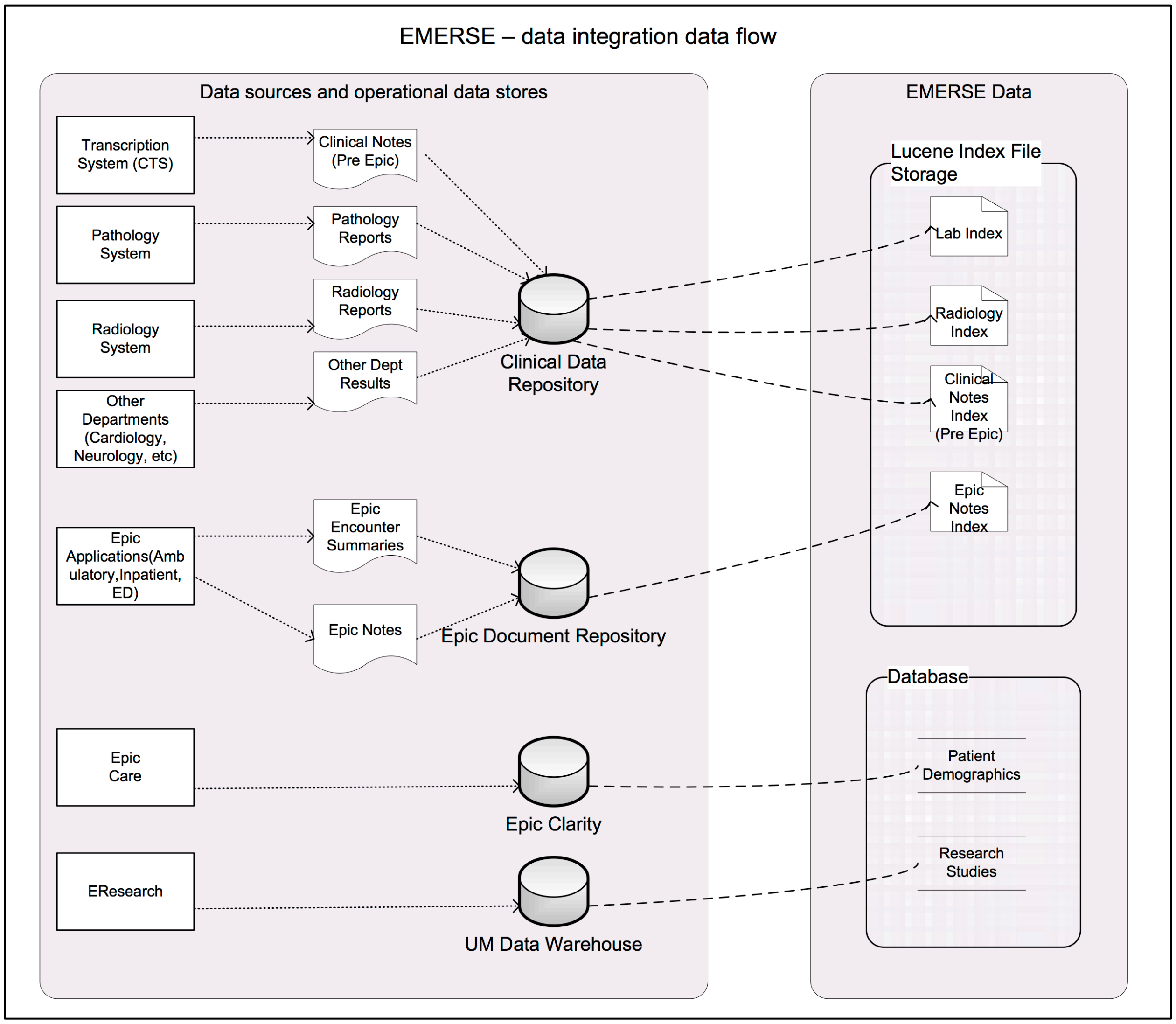


Figure. High-level overview of how data are integrated from various source systems into EMERSE at the University of Michigan. Each institution will likely have its own unique set of sources and integration issues.

Indexing using Solr DIH (Data Import Handler)

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

The easiest way to index documents stored in your local document store database is to use the DataImportHandler (DIH) that ships with Solr. 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 to be placed in the lib directory of Solr installation:

1. `solr-dataimporthandler-4.x.x.jar`
2. `solr-dataimporthandler-extras-4.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>
```

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_nostopwords" name="RPT_TEXT" indexed="true" stored="true" termVectors="true"
   termPositions="true" termOffsets="true" />
<field name="RPT_TEXT_NOIC" type="text_general_nostopwords_nolowercase" indexed="true" stored="true"
   termVectors="true" termPositions="true" termOffsets="true" />
<field name="_version_" type="long" indexed="true" stored="true"/>
```

A few things to note:

1. There are multiple ways to create indices 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_nostopwords` is an extension of Solr `text_general` wherein index and query analyzer does not use any stopwords filter

4. The field type `text_general_nostopwords_nolowercase` is an extension of `text_general _nostopwords` where lowercase filter is skipped in addition to stopwords

<u>Indexing programmatically with SolrJ</u>

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.

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

Solr connects to the document repository using SQL/JDBC. We are using a Unix Cron job to tell the indexing code to start and run once per night vian HTTP call to the running app.

Updating the 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 (~90 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 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.

Interactions with an EHR like Epic

The University of Michigan Health System (UMHS) EHR is based on Epic. EMERSE receives encounter summaries and notes from Epic via an Epic Bridges outbound HL7 interface. Details about this interface can be found here:

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

The emitted HL7 messages include the clinical notes in RTF format, which preserves the formatting and structure of the notes. These files are converted to HTML by ASPOSE Words API, and stored for nightly indexing into Lucene format.

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

EMERSE is designed to run stand alone, so there are no real-time Web Service calls to Epic. It is also possible to receive plain text notes either through HL7 or through 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 if that were 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.

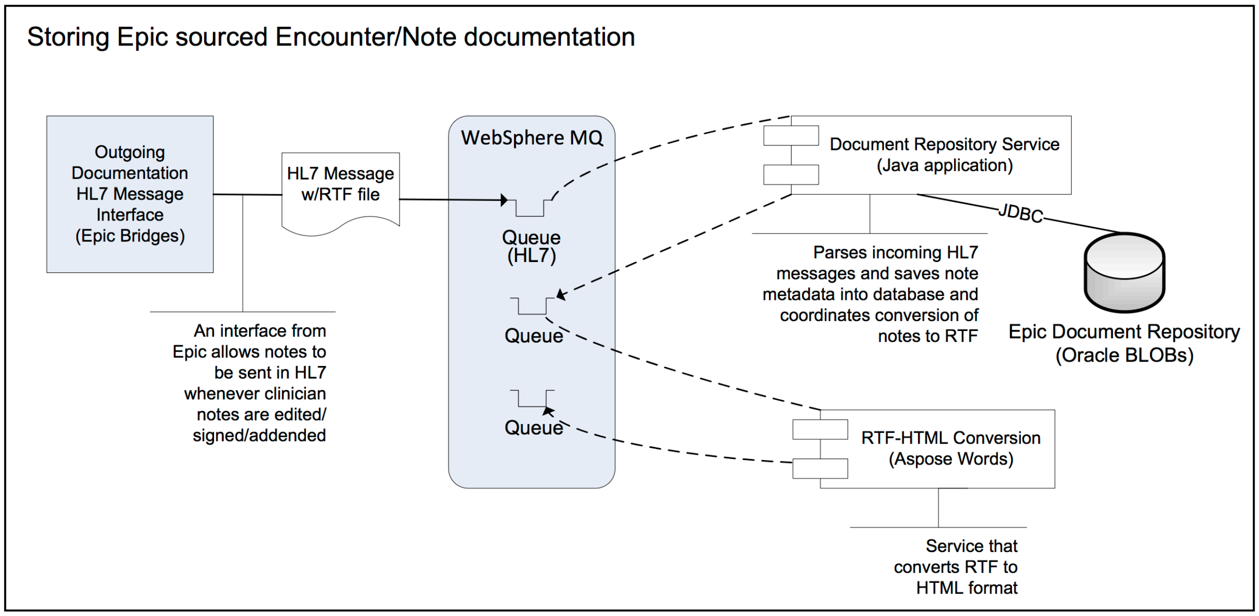

Storing Epic sourced Encounter/Note documentation
Outgoing Documentation HL7 Message Interface (Epic Bridges)
HL7 Message w/RTF file
An interface from Epic allows notes to be sent in HL7 whenever clinician notes are edited/ signed/addended
WebSphere MQ
Queue (HL7)
Queue
Queue
Document Repository Service (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

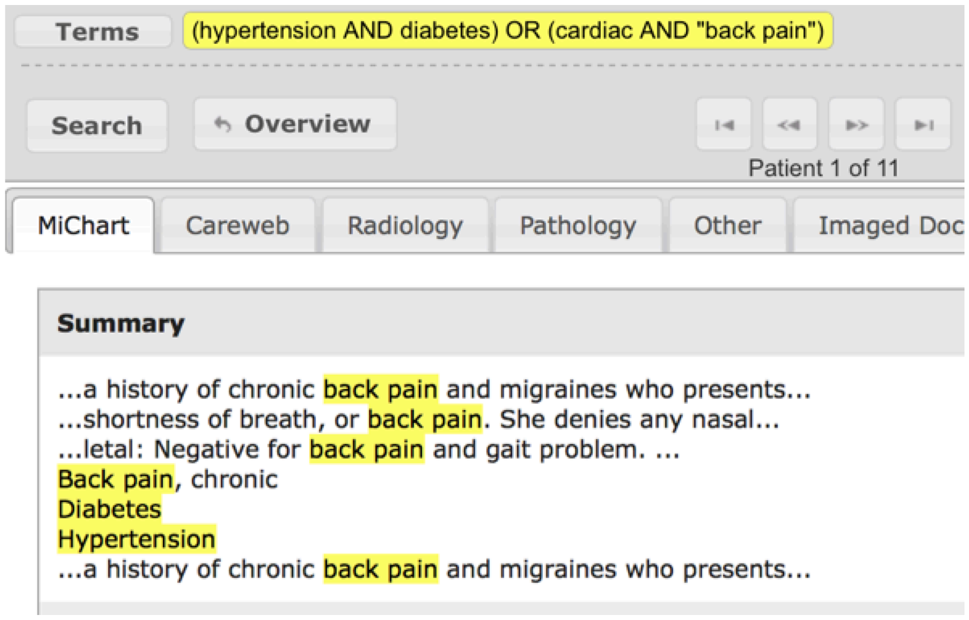
Figure. The approach used at the University of Michigan to obtain notes from the Epic EHR and make them searchable within EMERSE.

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.



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...