

# CS144: Web Applications

## Project Part 3: Search and Retrieval

**Due date:** Friday, February 17, 2017, 11:00 PM

### Important Notes

- **Submission deadline:** Programming work is submitted electronically and must be submitted by Friday at 11:00 PM. However, we recognize that there might be last minute difficulty during submission process, so as long as you started your submission process before 11:00PM, you have until 11:55PM to completely upload your submission. After 11:55PM, you will have to use grace period as follows.
- **Late Policy:** Programming work submitted after the deadline but less than 24 hours late (i.e., by Saturday 11:00 PM) will be accepted but penalized 25%, and programming work submitted more than 24 hours but less than 48 hours late (i.e., by Sunday 11:00 PM) will be penalized 50%. No programming work will be accepted more than 48 hours late. Since emergencies do arise, each student is allowed a *total* of four unpenalized late days (four periods up to 24 hours each) for programming work, but no single assignment may be more than two days late.
- **Honor Code reminder:** For more detailed discussion of the Honor Code as it pertains to CS144, please see the Assigned Work page under [Honor Code](#). In summary: You must indicate on all of your submitted work *any assistance* (human or otherwise) that you received. Any assistance received that is not given proper citation will be considered a violation of the Honor Code. In any event, you are responsible for understanding and being able to explain on your own all material that you submit.

---

### Overview

In Part 2 of the project, you designed a relational schema for a snapshot of eBay auction data provided in a set of XML files. In this part you will provide "search" functions to this data. Your search functions will support two types of queries: keyword-based search over text fields (such as item descriptions) and search for items located within a geographic region. For the first type of queries, you will be using the [Apache Lucene](#) text search engine. For the second type, you will use a spatial index in MySQL.

In Part A, you will implement a basic keyword search functionality that allows searching for auction items based on a set of keywords. This will be done by reading data from MySQL and creating an inverted index with Lucene library. In Part B, you will then implement a spatial search functionality that allows searching for auction items based on keywords and their locations. To support efficient spatial queries, you will need to create a spatial index in MySQL and use the index to filter items by their latitude and longitude.

By completing this project you will be able to learn:

1. How to use JDBC to access a database in Java
2. How to use Lucene to build and use an inverted index
3. How to create a spatial index in MySQL

We will be providing relevant tutorials and references to help you learn the above topics.

## Part A: Keyword Search

In Part A, you need to implement the following basic keyword search functionality over eBay data.

- Basic keyword search: return the itemId and name of all items that contain a set of keywords. The search should be performed over the *union* of the item name, category, and description attributes.

For example, for the query "Disney", your basic search function should return the itemId and name of items that have the keyword "Disney" in the *union* of the name, category or description attributes. Also, for multiple word queries, such as "**star trek**", **you should consider that as "star" OR "trek"**. That is, you should return any item if it has either "star" or "trek" in the union of the keywords in the name, category and description attributes.

To support the basic search function, you will have to build an inverted index capable of searching over the union of the three attributes in your database schema, using an open-source library called Lucene.

### A.1: Learn About JDBC and Lucene.

In order to access MySQL data from a Java program, you will need to use JDBC (Java DataBase Connectivity) API. You will also use Lucene to build inverted indexes for text fields and search over them. To learn how to use JDBC and Lucene, first go over the following two tutorials.

- [A short tutorial on JDBC](#)
- [A short tutorial on Lucene](#)

Also, if you haven't gone over our [short tutorial on Ant](#) in Project 2, please do it now because we heavily rely on Ant for this part of the project. Proceed to the next step only after you have gone over the above tutorials.

### A.2: Write Java Application to Create Lucene Index

In order to support the basic keyword search function described before, you will first need to create a Lucene index on a number of attributes of your database tables. Decide which index(es) to create on which attribute(s). Document your design choices and briefly discuss why you've chosen to create your particular index(es) in a plaintext file called `README.txt`.

Once you have chosen what Lucene index(es) to create, write a Java application to create them, iterating through your database tables to add the appropriate index entries to the index(es). As a starting point, we have provided skeleton code `DbManager.java` and `Indexer.java` in the `src` directory of [project3-indexer.zip](#).

Note that the two Java classes are defined as part of the `edu.ucla.cs.cs144` package, so the java files are located at `src/edu/ucla/cs/cs144` to be precise. If you are not clear about how a package name is related to the source directory, read Section 1.8 of [A Crash Course from C++ to Java](#) again. As part of your implementation, you will probably want to create new Java classes for modular development. If you do, make sure that your new classes become part of `edu.ucla.cs.cs144` package and place your source files in the `src/edu/ucla/cs/cs144` directory.

The provided `DbManager` class encapsulates the database connection information for JDBC, so that the information is separate from your main code. For any reason, if you decide not to use this class, make sure to use the database `cs144` and connect it as the user "`cs144`" with empty password. The `Indexer` class contains the bare minimum code for your Lucene index creation module. Inside this class, you will have to use JDBC to retrieve Item information from MySQL and use Lucene `IndexWriter` class to create Lucene indexes. Again, if you are not familiar with how to use JDBC and Lucene API, read [JDBC tutorial](#), and [Lucene tutorial](#).

When you create your Lucene index(es), **make sure to create them under `/var/lib/lucene/` directory**. For example, if you want to create your (first) Lucene index in the `index1` subdirectory, it should be created at `/var/lib/lucene/index1/`.

The `project3-indexer.zip` file contains ant `build.xml` file, which has the target "compile" to compile your Java source code and the target "run" to invoke `edu.ucla.cs.cs144.Indexer` for indexing. If needed, update your `build.xml`, so that the Ant target "run" always executes your Lucene indexing module and builds the index(es).

**Notes on MySQL and Lucene library files:** As we explained in the JDBC and Lucene tutorials, if you want to use MySQL JDBC and Lucene libraries in your code, you need to pass the location of the two jar files (`mysql-connector-java.jar` and `lucene-*.jar` in `/usr/share/java`) through the `classpath` option of `javac` and `java`. Otherwise you are likely to get "class not found" error. Since the provided `build.xml` file sets the `classpath` option automatically, you are unlikely to encounter this problem for this project as long as you use our `build.xml` file.

### A.3: Implement Basic Keyword Search Function

Now that you have created a (set of) Lucene index(es), you will implement the basic search functionality. In particular, you will need to implement the `basicSearch` function of the following Java interface (`spatialSearch` and `getXMLDataForItemId` functions will be implemented later in Part B):

```
public interface IAuctionSearch {
    public SearchResult[] basicSearch(String query, int numResultsToSkip, int
numResultsToReturn);
    public SearchResult[] spatialSearch(String query, SearchRegion region, int
numResultsToSkip, int numResultsToReturn);
    public String getXMLDataForItemId(String itemId);
    public String echo(String message);
}
```

You'll find this interface definition in the `IAuctionSearch.java` file located in the `src/edu/ucla/cs/cs144` directory of [project3-searcher.zip](#). Roughly, given a query, `basicSearch()` should return `numResultsToReturn` items after skipping the top `numResultsToSkip` matching items. For example, if `numResultsToSkip = 10` and `numResultsToReturn = 20`, the functions should return top-11 through top-30 matching auction items as the result. Read `IAuctionSearch.java` to understand what each method should do, which is described as a comment within the file. The `SearchResult.java` file has the definition for the `SearchResult` class, which is used by the `basicSearch` method to return search results. It has two attributes, `itemId` and `name`, that are the id and the name of a returned eBay item.

The sample code `AuctionSearch.java` in the same directory of the zip file has the `echo()` method implemented for you. You will have to add your code to the parts marked as "TODO: Your code here!". To identify the items that contain the query keywords, your implementation of `basicSearch()` method must use Lucene index(es) that you built in Part A.2, which is located below `/var/lib/lucene/`. The search should be performed over the *union* of the item name, category, and description attributes and your method should return (id, name) pairs of matching items.

**Note:** You may use any ranking method for ordering the results from `basicSearch`. Lucene's default ranking may be a good choice, for example.

**Java String comparison:** To compare Strings in Java, you must use the `equals` method, because the operator `==` in Java compares object references not the object values.

To help you debug your `basicSearch()` methods, we provide some sample queries and the number of matching items.

- "superman": 68 matches
- "kitchenware": 1462
- "star trek": 770

**Note:** In order to get the above results, you have to assign datatype **`varchar(4000)`** to the **Description** field, as we suggested in Project 2.

## Part B: Spatial Search

In this part, you will implement two remaining methods of `IAuctionSearch` interface: `spatialSearch()` and `getXMLDataForItemId()`.

```
public interface IAuctionSearch {
    public SearchResult[] basicSearch(String query, int numResultsToSkip, int
numResultsToReturn);
    public SearchResult[] spatialSearch(String query, SearchRegion region, int
numResultsToSkip, int numResultsToReturn);
    public String getXMLDataForItemId(String itemId);
    public String echo(String message);
}
```

The `spatialSearch()` method is an extension of the `basicSearch()` method and it returns items if they satisfy both of the following two conditions:

1. The items should have at least one keyword in the query parameter in their title, category, or description fields (this is exactly the same condition used in `basicSearch()`) and
2. The items should be located within the rectangular region specified in the `region` parameter. A region is specified with four coordinates, `lx`, `ly`, `rx`, and `ry`, where (`lx`, `ly`) corresponds to (latitude, longitude) of the lower left corner of the rectangle and (`rx`, `ry`) to (latitude, longitude) of the upper right corner. The `SearchRegion` class is defined in the `SearchRegion.java` file located in `src/edu/ucla/cs/cs144`.

We will explain the `getXMLDataForItemId()` method later.

## B.1: Learn about Spatial Index Support in MySQL

You can use the Lucene index(es) that you built earlier to check the keyword-matching condition of `spatialSearch()`. To check the spatial condition, it is possible to build two separate B+tree indexes on latitude and longitude attributes and look up both indexes using the specified region. This approach is likely to be better than a full scan of the entire table(s), but a better approach is to build a spatial index that indexes on both attributes simultaneously and intelligently divides the two dimensional space, so that items within a given region can be identified much more efficiently with a single index lookup.

In MySQL, it is possible to create a spatial index (in particular, R-tree index) on geometric (or spatial) attributes for efficient spatial-query support. Go over MySQL manual pages on [Using Spatial Data](#). Pay particular attention to the section on [Creating Spatial Indexes](#) and [Using Spatial Indexes](#) to learn how to create and use a spatial index in MySQL.

## B.2: Build Spatial Index in MySQL

Now that you have learned how to use a spatial index in MySQL, it is time to create one. Note that in MySQL 5.5, a spatial index can be built only for MyISAM tables only, so we will first need to create a table that contains information on items' latitude and longitude using the MyISAM storage engine. In particular, the table should contain (itemId, latitude, longitude) of each item, where (latitude, longitude) is stored as a *single* column of POINT type so that a spatial index can be created on the column.

Write an SQL script that (1) creates such a table using MyISAM storage engine, (2) populate the table with itemId, latitude, and longitude information, and (3) create a spatial index on latitude and longitude. You can use [POINT\(.\)](#) function in MySQL to convert a pair of numeric values to a POINT. Name the SQL script as `buildSQLIndex.sql`. Run this script once, to create the table and the spatial index.

Write another SQL script containing the DROP TABLE and DROP INDEX statements that delete the tables and indexes for spatial query support. Name this script as `dropSQLIndex.sql`. (You MUST NOT run this script! You will need the spatial index for the next step.)

**Note:** Remember that MySQL automatically creates an index on the primary key of each table, so you will not have to create an index on the primary key of any table.

## B.3: Implement spatialSearch Method

Now implement the `spatialSearch()` method in the `AuctionSearch.java` file. Your implementation of `spatialSearch()` must use Lucene index(es) to handle the keyword constraint and MySQL spatial index for spatial constraint. If you followed our instruction in Part A, your Lucene index(es) must be located below `/var/lib/lucene/`. For querying and retrieving data from MySQL, make sure that your code uses the database `CS144` and connect it as the user "`cs144`" with empty password.

**Note:** You may use any ranking method for ordering the results from `spatialSearch`.

## B.4: Implement `getXMLDataFromItemId` Method

The third `getXMLDataFromItemId()` method of `IAuctionSearch` should return the XML-formatted item data as string given its `itemId`. Make sure that the output from `getXMLDataFromItemId()` conforms to the DTD of the original eBay XML data, except that the root element should now be `<ITEM>`, not `<ITEMS>` (note that this method returns only one eBay item, not multiple items). Also make sure that the format of each XML element is as close to the original XML data as possible, particularly if the SQL data type is `TIMESTAMP`. You will have to escape the character "<" to "&lt;" and ">" to "&gt;", and so on ([Reference](#)) if they are not part of an XML tag. Finally, if there is no matching Item for a given `itemId`, return an empty string.

**Note:** You can use the Java class [SimpleDateFormat](#) to parse and format date strings.

## Testing Your Code

We have included a simple test file `AuctionSearchTest.java` in `project3-searcher.zip`. You can use the ant target "run" to invoke this `AuctionSearchTest` class for basic testing of your code. Before you submit your code, add more test cases to `AuctionSearchTest`, thoroughly test your code, and make sure that your implementation is correct.

## What to Submit

For this project, you need to submit **a single zip file contains three sub zip files**, one for index construction, one for search and Web service deployment, and one for resubmission of your `project2.zip` (even if you do not make any changes).

## Indexer Submission

The first zip file, **project3-indexer.zip** that has the following packaging structure.

```
project3-indexer.zip
|
+- team.txt
|
+- buildSQLIndex.sql
|
+- dropSQLIndex.sql
|
+- build.xml
|
+- README.txt
|
+- src
|   +- java source codes for indexer (with your own naming/structure)
+- lib
    +- external java libraries used (that are not available on our VM)
```

Each file or directory is as following:

1. **team.txt**: A plain-text file (no word or PDF, please) that contains the UID(s) of every member of your team. If you work alone, just write your UID (e.g. 904200000).  
If you work with a partner, write both UIDs separated by a comma (e.g. 904200000, 904200001). **DO NOT put any other content, like your names, in this file!**
2. Your `buildSQLIndex.sql` and `dropSQLIndex.sql` that are used for creating and dropping an spatial index within MySQL.
3. Your `build.xml` file, the ant script for Lucene index construction. Make sure that the target "run" of your `build.xml` executes your Indexer and creates all of your Lucene index(es) below `/var/lib/lucene/`. *Do not submit the binary version of your Indexer or your created Lucene index(es)*
4. A **README.txt** any comments you find worth noting, regarding your code structure, etc.
5. Indexer source code should be in `src` directory.
6. Non-standard java libraries that you used and not available on our VM should be in `lib` directory.

## Testing of Indexer Submission

To ensure the correct packaging of your submission, we have made a grading script [p3\\_index\\_test](#) for indexer submission, which can be executed like:

```
cs144@cs144:~$ ./p3_index_test project3-indexer.zip
```

(Add the appropriate path to the `project3-indexer.zip` if needed. You may need to use "chmod +x `p3_index_test`" if there is a permission error.) The script assumes that your CS144 database already contains the tables that you created and loaded in Project 2. If not, run your project 2 submission first before you test this submission.

You **MUST** test your submission using the script to minimize the chance of an unexpected error during grading. When everything runs properly, you will see an output similar to the following from the script:

```
Dropping existing SQL index from dropSQLIndex.sql...
Creating SQL index from buildSQLIndex.sql...
Removing existing lucene index...
Running 'ant run' to create a lucene index...

... output from ant ...

Finished creating lucene index
This is the content of the lucene index directory:
... your index name(s) ...

Finished checking Project 3A submission
Please check the output of this script to ensure a proper submission
```

## Searcher Submission

The second zip file, **project3-searcher.zip** should have the following packaging structure.

```
project3-searcher.zip
|
+- build.xml
|
```



```

+- README.txt
|
+- src
|   +- java source codes for search (with your own naming/structure)
+- lib
    +- external java libraries used (that are not available on our VM)

```

Each file or directory is as following:

1. Your **build.xml** file for your searcher class. Make sure that the target "run" of your build.xml compiles your Java classes and executes AuctionSearchTest class to test your implementation. *Do not submit your compiled Java classes.*
2. A **README.txt** any comments you find worth noting.
3. All your Java source code files that implement IAuctionSearch interface, in **src** directory.
4. Any external java libraries that you used should be in **lib** directory.

Make sure that your scripts access other files (such as source codes, library dependencies, and resulting output files) properly with *relative paths*.

## Testing of Your Submission

Download the grading script [p3\\_searcher\\_test](#) for this submission, and test your zip file by executing:

```
cs144@cs144:~$ ./p3_searcher_test project3-searcher.zip
```

(Add the appropriate path to project3-searcher.zip if necessary. You may need to use "chmod +x p3\_searcher\_test" if there is a permission error.) The script assumes that your CS144 database has been populated with your tables and data from project 2 and your Lucene index has been created at /var/lib/lucene/ from project 3 part A. If not, run your project 2 and project 3 indexer submissions with their grading scripts before testing this submission.

You **MUST** test your submission using the script to minimize the chance of an unexpected error during grading. When everything runs properly, you will see an output similar to the following from the grading script:

```

Running 'ant run' to build your searcher and run test...
Buildfile: /tmp/p3b-grading/build.xml

init:
    [mkdir] Created dir: /tmp/p3b-grading/bin

compile:
    [echo] project3-searcher: /tmp/p3b-grading/build.xml
    [javac] Compiling 6 source files to /tmp/p3b-grading/bin

run:
    [java] ...

    ... output from AuctionSearchTest ...

    [java] ...

```



```
BUILD SUCCESSFUL
Total time: 1 second
```

```
Finished checking your Project 3B submission
Please check the output of this script to ensure a working submission
```

## Final Zip Submission Archive

Please create a submission archive named **project3.zip**, with the following structure:

```
project3.zip
|
+- project3-indexer.zip
|
+- project3-searcher.zip
|
+- project2.zip
```

That is, we require you to submit your work for **project3 together with project2 (again)** via our submission page at [CCLE](#), which means **you are allowed to make changes to project2** if it hinders the correct implementation of project3. Your project2 score will not change even if you find and correct the error in this resubmission, and you should always make this resubmission regardless of whether you make changes or not.

The three zip files should be **contained directly** under the **project3.zip** (without any enclosing folders).

You may submit as many times as you like, however only the latest submission will be saved, and those are what we will use for grading your work and determining late penalties.

## Grading Criteria

### Breakdown

- Your java code compiles and runs fine (10%)
- `basicSearch()`: test whether Lucene index is created and used correctly (30%)
- `spatialSearch()`: test whether Lucene/MySQL index are created and used correctly (30%)
- `getXMLDataForItemId()`: test whether XML data is constructed correctly (30%)

### Correctness vs Efficiency

We grade your project based on the "correctness" of your work. Efficiency of your work is not a primary concern, but your codes **MUST** complete in a "reasonable" time:

- Creating index (Lucene and MySQL): should complete in 20-30 seconds. No penalty if your work completes in 2 minutes; after that, 20% off of your entire project score. Your code will be considered as a failed work if it takes 5 minutes or longer.
- Answering one basic search AND one spatial search: should complete in 15-20 seconds. No penalty if

your work completes in 1 minutes; after that, 20% off of your entire project score. Your code will be considered as a failed work if it takes 3 minutes or more.

- Note: the above penalties are cumulative.