# Temporary New Interface Title

Aditya Kaulagi, Gabe Fierro
Coleman Fung Institute for Engineering Leadership
UC Berkeley
aditya15@berkeley.edu, fierro@eecs.berkeley.edu

December 9, 2013

## Abstract

In this document, I describe the process of constructing SQL Queries from a HTML form, which are used to get results from a database. These results are then emailed to the person who requested them.

## 1 Introduction

There was a big database which had a lot of information related to patents. This database was divided into tables each of which contained different parts of information (patent information, inventor information, assignee information, lawyer information, etc.). However, before creating this interface, the only way to access this information was by making SQL queries. This required login credentials to the MySQL database, and SQL query forming expertise. To give a person without these privileges access to this information, we needed a web page where users could select what information they want from this database.

The goal was to make a user friendly web interface which allowed users to select what rows they wanted from the tables in the database, specify any filters they wanted to specify for these rows, enter their email address, and specify the format in which they wanted the information (CSV, TSV, or SQLITE3). To do this, we selected to use the Django[1] framework because it was easy to install and learn. And then, to run the query itself, we used an external python file that imported the Django settings and just ran jobs if any were available (or sleep for some specific amount of seconds if no jobs were available). Our final goal was to make an app that would do this sort of forms to queries transformation on any database with minimal changes to the files included in the app.

## 2 Structure of the Application

The application is available on github at `https://github.com/gtfierro/walkthedinosaur`. The application consists of three services: the django app that parses user input and turns it into a query, a python file that executes the query, gets the result from the remote MySQL database, writes it to a file and emails the user a notification which contains a download link to the file, and a fileserver that serves files to the user.

All the models, templates and views are stored in the batchsql folter. The python file that executes the query is called `run_jobs.py`. The information to connect to the remote server and other configurations are stored in `config.ini`.

## 3 Structure of Database

The database[1] is divided into many tables. Out of these tables, the tables we can currently get information from (and their columns) are:

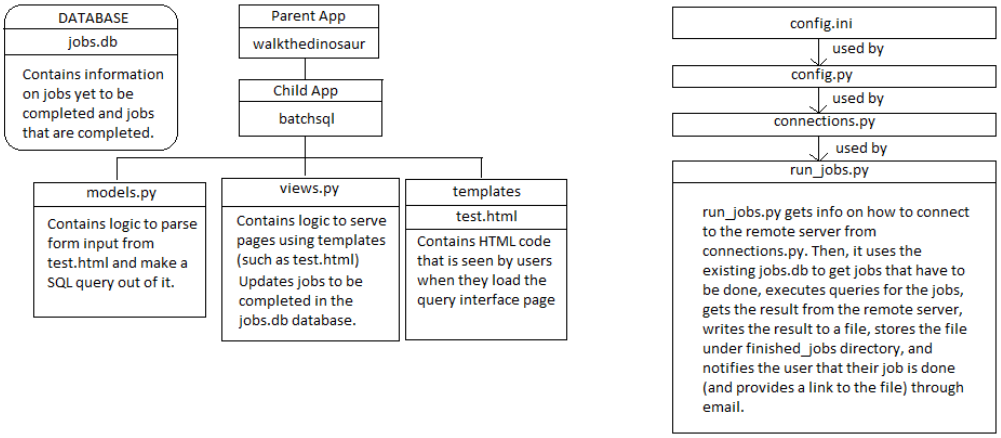


Figure 1: Structure of app

1. `patent`: This table contains primary information about the patent.
   (a) title: The Title of the Patent
   (b) id: The patent ID. This is a unique number for each patent.
   (c) date: The date on which this patent was issued.
   (d) country: The country where this patent was issued.

2. `rawinventor`: This table contains information about the patents inventor.
   (a) name_first: The first name of the inventor.
   (b) name_last: The last name of the inventor.
   (c) nationality: The nationality of the inventor.
   (d) location_id: Column to establish a relationship between inventors and locations. This contains the id of the location of the inventor.
   (e) patent_id: Column to establish a relationship between a patent and its inventor. This contains the id of the patent to which it refers to.

3. `rawassignee`: This table contains information about the entity that assigned this patent.
   (a) name_first: The first name of the assignee.
   (b) name_last: The last name of the assignee.
   (c) nationality: The nationality of the assignee. organization: Name of the organization this assignee belongs to (if any).
   (d) location_id: Column to establish a relationship between inventors and locations. This contains the id of the location of the assignee.
   (e) patent_id: Column to establish a relationship between a patent and its assignee. This contains the id of the patent to which it refers to.

4. `rawlawyer`:
   (a) name_first: The first name of the inventor.
   (b) name_last: The last name of the inventor.
   (c) organization: Name of the organization this assignee belongs to (if any).

(d) country: Country where the lawyer (organization) exists.

(e) patent_id: Column to establish a relationship between a patent and its lawyer. This contains the id of the patent to which it refers to.

5. `rawlocation`: This table stores information for different locations.

   (a) id: Unique number for each location.

   (b) city: The name of the city.

   (c) state: The name of the state.

   (d) country: The name of the country.

6. `claim`: This table stores information about the claims of every patent.

   (a) patent_id: Column to establish a relationship between a patent and its lawyer. This contains the id of the patent to which the claim refers to.

   (b) text: The text of this claim.

   (c) dependent: ID of the claim this claim is dependent on.

   (d) sequence: ID of this claim.

7. `uspatentcitation`: This table stores information on all the citations in a patent.

   (a) patent_id: Column to establish a relationship between a patent and its lawyer. This contains the id of the patent to which it refers to.

   (b) date: The date this citation was made.

   (c) country: The country in which this citation was made.

   (d) sequence: The ID of the citation.

# 4 Converting HTML form to SQL Query

All of the conversion from HTML form to a SQL query is done in `batchsql/models.py`. All of the form variables are given to the TestQuery class through the post variable that we get from django. In post, all the values entered by the user are stored as a dictionary in the form fieldname:value. All form elements are broadly categorized into three types:

1. Field Variables: These are the columns that the user wants information from. For example, Name of a Patent, or Name of the Inventor of the Patent.

2. Filter Variables: These are the filters specified by the user. They are mostly textboxes or select lists. If a user enters TX under the Inventors location filter, then all the rows (in the columns specified by field variables) that have the inventors location as TX will be returned.

3. Miscellaneous: The csrf token, the email address, the file type that the user wants the information in are considered as miscellaneous fields as models.py does not use these fields to make queries.

In models.py, we have a dictionary which maps the form elements names to (table,column) which they represent. For example, the Patent Title represents the patent table and the title column, and hence one of the entries in this dictionary will be `pri-title:(patent, title)`(where pri-title is the name of the field for Patent Title). All fields have a prefix of f to separate them from filters. Converting the form elements is a 4 step process:

1. Get columns that the users want in their results and store it in a set. This is generated from the Field Variables.

2. Get the names of tables to be searched and store it in a set. This is generated from both the Field Variables and Filter Variables.

3. Get the filter conditions and store it in a set. This is generated from both the Field Variables (for cross-referencing between tables) and Filter Variables.

4. Loop through the above sets and construct a query of the structure

SELECT {table.columns} FROM {tables}
WHERE {filters};

Once this query is generated, it is stored in a local databse that stores the queued and completed job information, and then the `run_jobs.py` file gets this query from this database and runs the jobs that have not yet been completed. It uses sqlalchemy[2] to connect to and execute queries at the remote MySQL[3] database.

# 5 Example Usage

## 5.1 Example 1

Lets say one needs to get the title of all the patents that had been invented in Texas between the period January 2005 and February 2005. To do this, perform the following steps (screenshots shown after the steps):

1. Select the checkbox besides Title of Patent in primary information.

2. In the filters section, under Primary information, set From as 2005-1-1 and To as 2005-2-1. Also, type TX in inventors state textbox.

3. Finally, type in your email address on the bottom of the page, choose the filetype, and click on Submit.

This form is translated into the SQL query:
SELECT patent.title FROM patent, rawinventor, rawlocation WHERE (patent.date BETWEEN 2005-1-1 AND 2005-2-1) AND (patent.id = rawinventor.patent_id) AND ((rawlocation.state LIKE %TX%) AND rawlocation.id = rawinventor.rawlocation_id);

## 5.2 Example 2

For the second example, lets say one needs to get the names (first and last) of the lawyers that filed for a patent in Michigan between January 2005 and February 2005. In addition, say they want the inventors and assignees to also be in Michigan[5].

1. Select First Name of Lawyer and Last Name of Lawyer under Lawyer Information.

2. In the filters section, make sure to fill in dates as before, and this time, fill in the textbox for Inventors State with MI and same for the Assignees State textbox.

3. Finally, just as before, fill in your email, choose your filetype, and click on Submit.

This form is translated into the SQL query:
SELECT rawlawyer.name_first, rawlawyer.name_last FROM patent, rawlocation, rawinventor, rawassignee, rawlawyer WHERE (patent.date BETWEEN 2005-1-1 AND 2005-2-1) AND (patent.id = rawinventor.patent_id) AND (rawassignee.patent_id = rawinventor.patent_id) AND (rawlawyer.patent_id = rawinventor.patent_id) AND ((rawlocation.state LIKE %MI%) AND rawlocation.id = rawinventor.rawlocation_id) AND ((rawlocation.state LIKE %MI%) AND rawlocation.id = rawassignee.rawlocation_id);

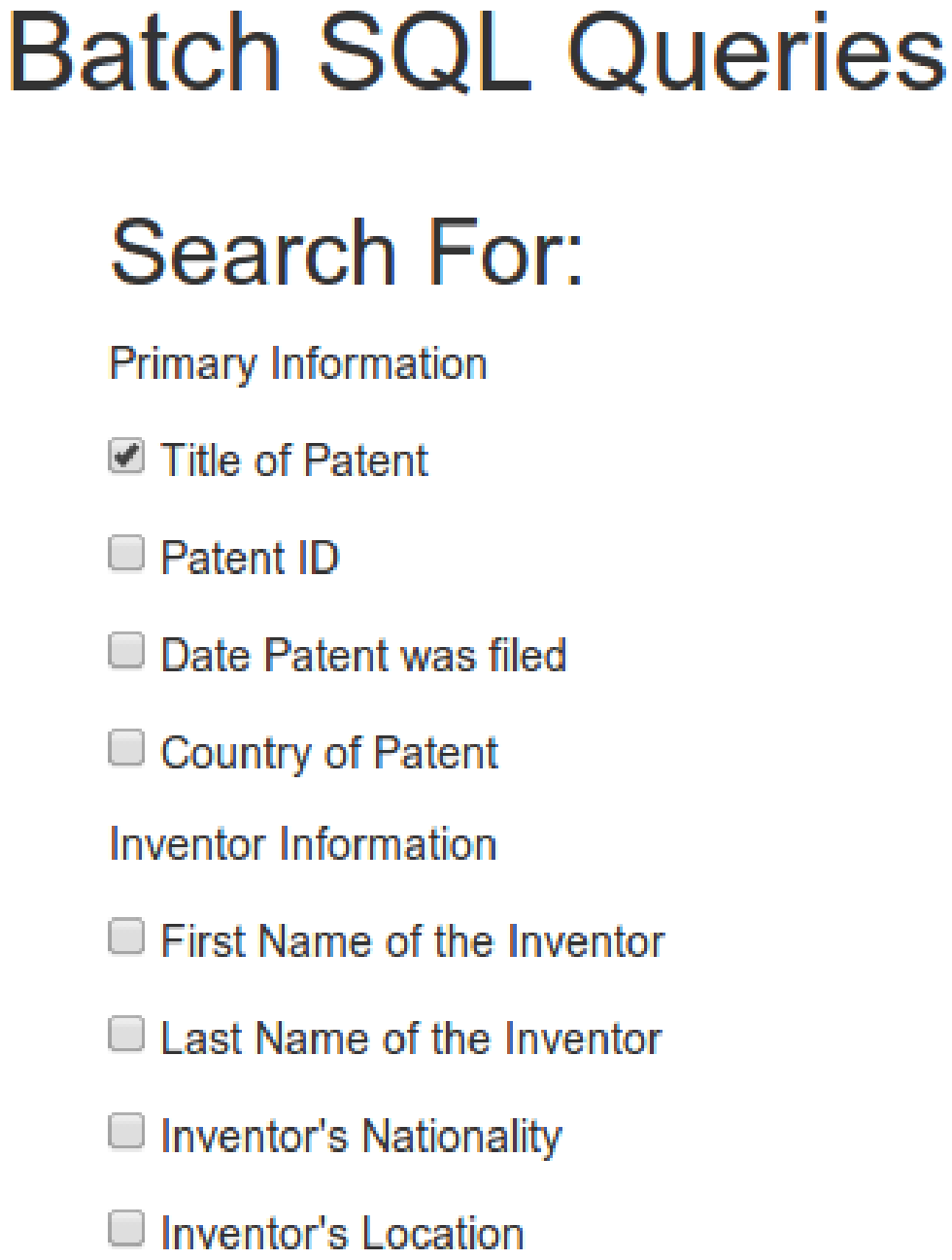


Figure 2: Example 1: Step 1

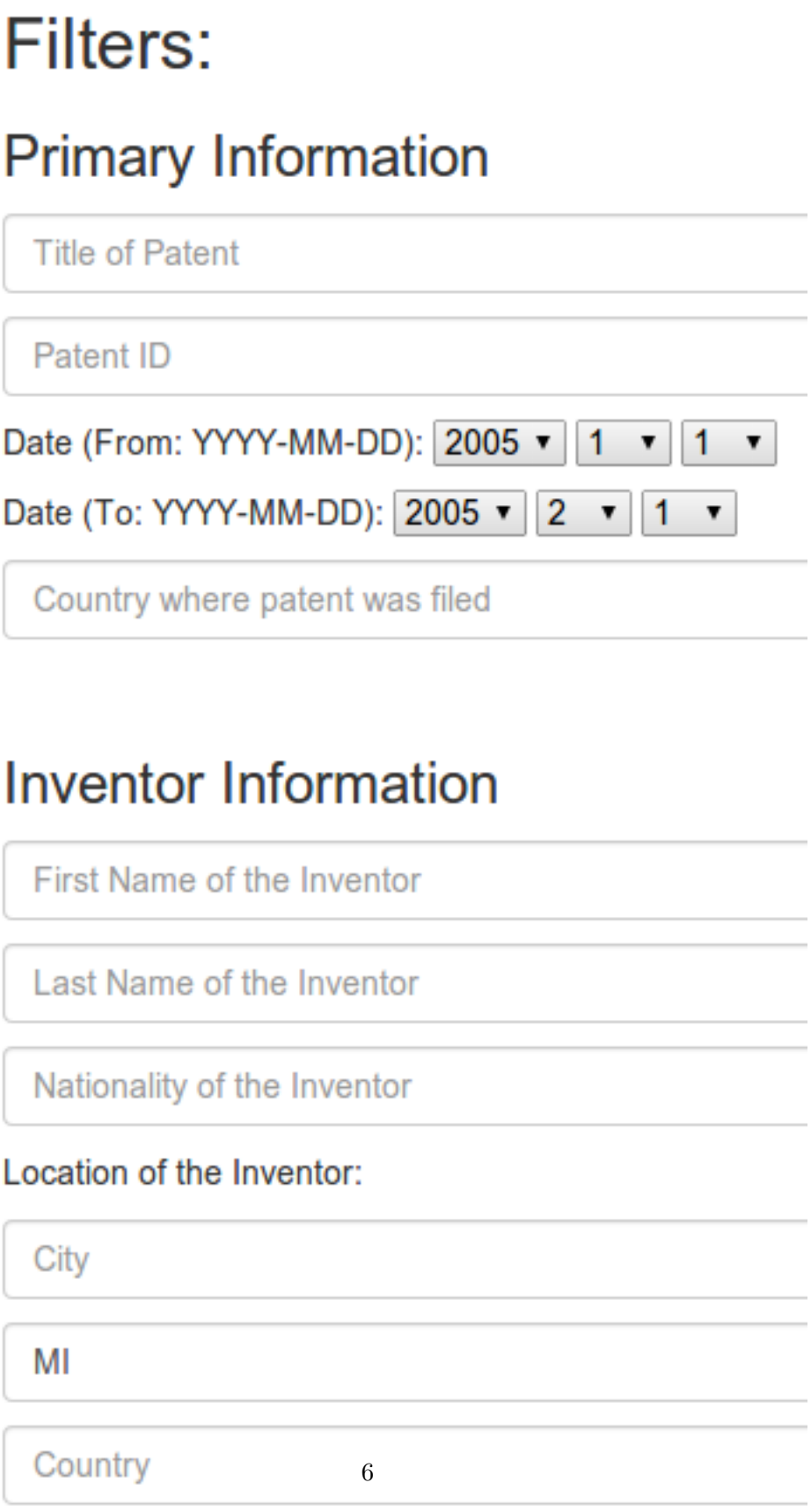




Figure 3: Example 1: Step 2

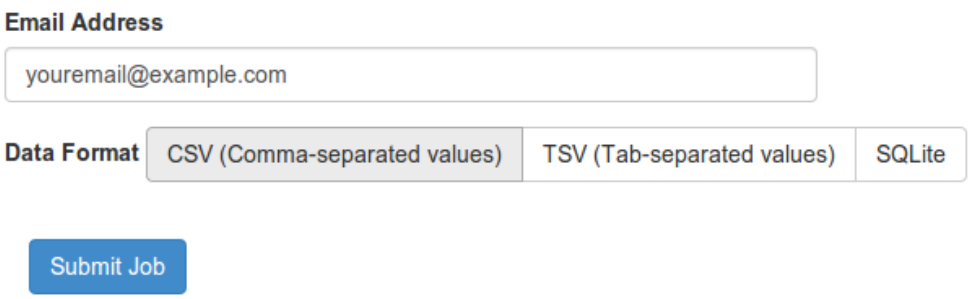


Figure 4: Example 1: Step 3

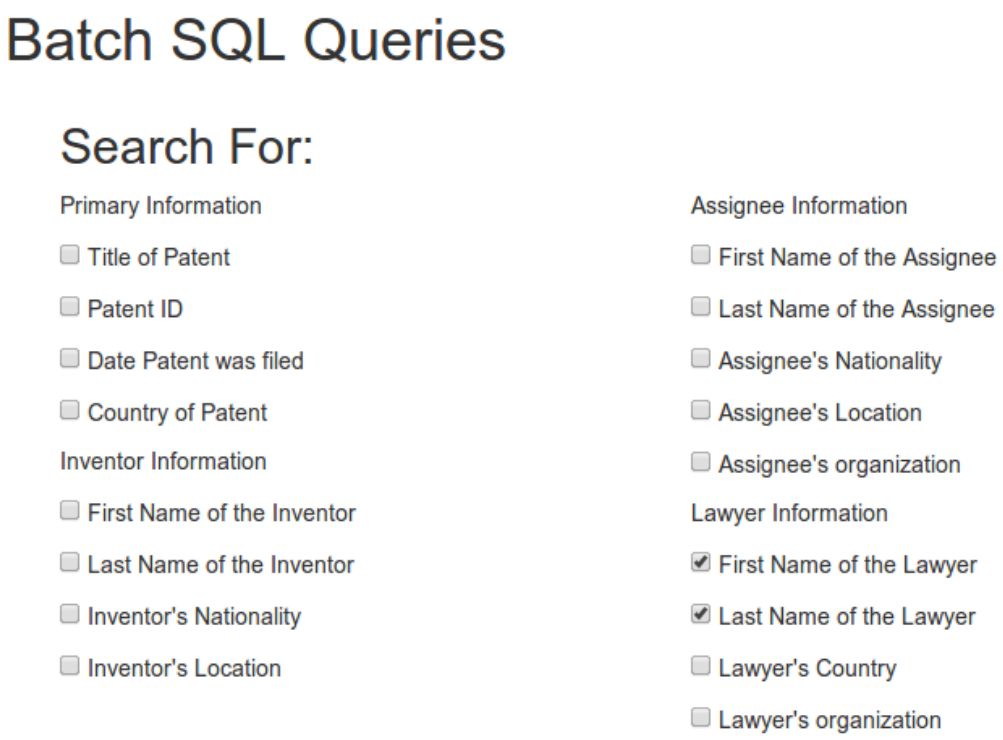


Figure 5: Example 2: Step 1

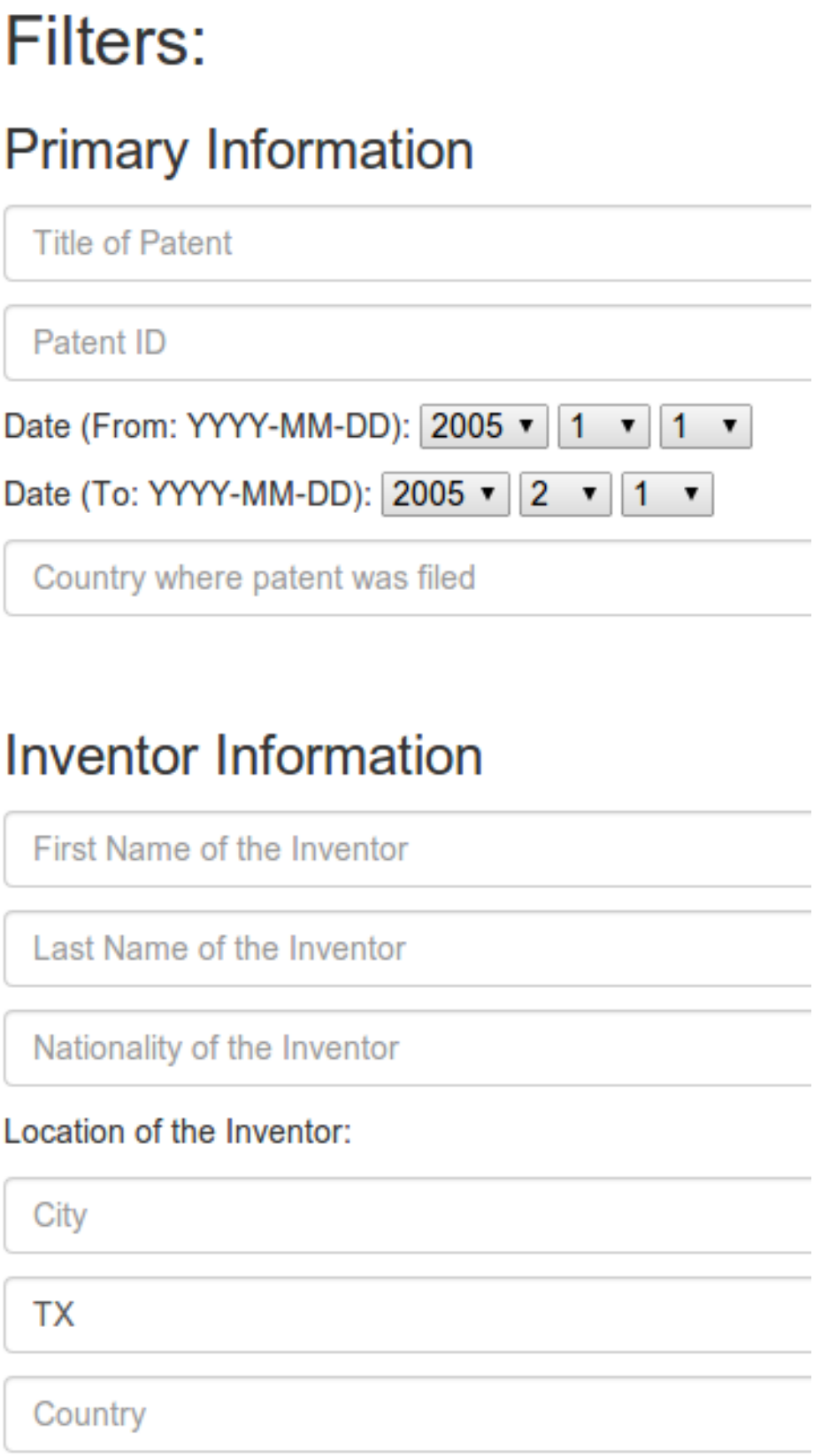




Figure 6: Example 2: Step 2-1

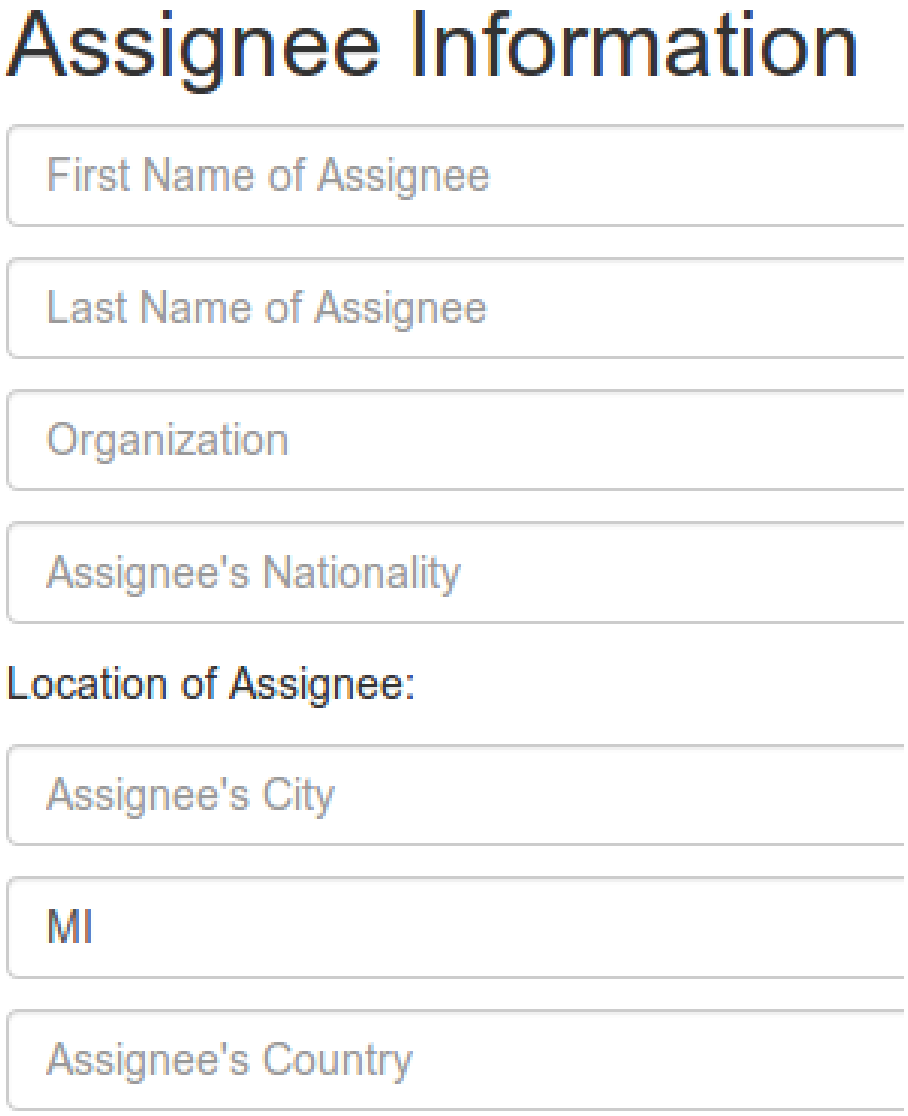


Figure 7: Example 2: Step 2-2