The webpage URL is: ***https://www.geos.ed.ac.uk/~s1757431/web_mapping.html***
This webpage has the best functionality on *Google Chrome* and *Safari* browsers.

Description of the web mapping project:

A FIELDS-FINDS archeological database is available. An interactive interface to this database is required. To this end, we need python codes to 1) query the Oracle database to retrieve the data in fields and finds tables. 2) do necessary computation and 3) display the results on a web browser.

Solution:

I used 1 html file, 1 java script file, and 7 python files to design a webpage with three different tabs and fill them with relevant information from database.

Web_mapping.html: This html file is the main file, which makes the general view for the each one of the following three tabs, also contains the address of related python files for these three tabs.

The first tab (Chart): shows the "*archeological finds map*". This map has the possibility of zooming in and out, also moving around when zoomed. Moreover, by clicking on each finds the relevant information will pop up in a proper window. By re-clicking that find the window will be closed.[1] Depending on the clicking position, I used if statements to position the windows dynamically to avoid them cutting the page's borders.

The second tab (Fields Table): shows the "*fields table*". This table is a copy of the fields table in the database. Using 'x' and 'add field' buttons on the page, user can remove an existing row or add a new row to the table. Any changes made by user will be saved on the copy version table (named fields2 on the codes) and will be shown by automatically refreshing the page. If user presses the reset button the primary values for the table will be again copied from the field (unchanged table) and the table will contain the default values. Domains for entering each value is set, in case of entering inappropriate input values by user, a message pops out saying the right domain values for the cell.

The third tab (Finds Table): shows the "*finds table*". This table has all the features explained in the second tab, except it is for finds table.

Python files:

Map.py
This file contains the content of the 'Chart' tab.
It consists of four functions:
get_text_tagStr: This function is responsible for creation of any text (string or number) in the grid tab.
get_field_tagStr: This function creates rectangles with the use of 'rect' SVG tag. It draws fields and the container of their id numbers in the middle of each field. get_text_tagStr is also called to fill the container with the corresponding id number.

[1] To make the ability of comparison possible I avoided to set time out for closing the windows, but the code for this purpose is commented in the map.py file.

get_finding_tagStr: It uses SVG circle tag to point the location of the finds on the grid tab. It also calls get_text_tagStr to allocate appropriate id number to each find.
get_legend_tagStr: It uses the rect and circle SVG tag to draw the legend for the map. Also calls get_text_tagStr to enter desired texts to legend section.
The needed input arguments for functions comes from the connection to the database.

Fields.py
It contains the content of fields tab. The get_field_row_table function makes one row of the table, using for loop for every column in the fields table. I connect to the database and by calling the get_field_row_table function for each table row, the whole rows for the html table is created.

Finds.py
It has the same procedure as the Fields.py except for the finds table values.

Credentials.py
In this file, I made a dictionary to save my real username and password as values for 'user' and 'password' keys for security purposes. Wherever I needed to login to the database in python codes, I imported the credentials library and used the values of 'user' and 'password' instead of the real ones. I also changed the mode for this file not to be accessible for others and groups. I did not include this file in my prints.

Delete.py
In this file I made two variables (fid, tab). Based on which tab(le) we are in (Finds or Fields), a database query is used to delete the row with the specific field/find id (fid). This python file will be called by ui.js file, when the user presses the 'x' button at the first of each row.

Insert.py
This file gets the new values from the user by post method. It uses insert query to put a new row with the corresponding values into the database, then reloads the page and shows the new row. The unique id is allocated to the row by finding the maximum of id(s) and increasing its value by one. This File is called by the ui.js file, when the 'add value' button in Finds or Fields tab is pressed.

Reset.py
This file does the automatic refresh of the webpage, when it is called in fields.py or finds.py. It connects to the database and deletes the content of copied version of fields or finds tables (fields2 or finds2) and inserts the default value amounts from untouched tables into copy ones.

Ui.js
It Contains java scripts, and as it was mentioned before, its functions are used in different python files.

How to improve to a full-blown web mapping system:

For future improvement, each user could have a username and password for logging into the system; this helps us with the controlling of the level of accessibility for each user. A real topographical map for the background of the Grid tab could be used. To prevent loading massive volume of data a default map (limit in the extent of the x and y coordinates) for each user could be allocated based on the user's interest and business status. The zooming and spanning ability could be improved, with consideration to load the neighborhood data on the map when spanning is stopped to prevent delays. The shape of the fields, which are assumed rectangles or squares,

could include more shapes like hexagonal (for specific needs). To make my webpage more user friendly I could allow users to drag two points to the table showing the diameter of the fields (instead of entering the coordinates manually). A full blown web mapping system should take the coordinates in one or more well-known coordinate systems and check if the inserted fields have overlap with other fields or not. More related tables from database could be joined to give users more information. Not every user should be able to add to the database or remove from it. Guidance notes should be added to help users working with the web page. More meticulous constraints and domains should be applied to the entering fields. The codes could be object oriented with classes for fields, finds, users, maps, tables, etc. Photos could be used to makes the webpage more colorful, and users could drag (for example) crops' image to the map instead of entering their names in the table. The pop up window on the map could have a close button on its self.

The content of all the codes are shown below (except credentials.py)

web_mapping.html

```
<!DOCTYPE html>
<head>
<title>Archaeological Finds Map</title>
<script>

// Styles for the active and inactive tables
var ACTIVE_TAB_STYLE = 'cursor: pointer; font-weight: bold; background: #dddddd;'+
'min-width: 80px;';
var INACTIVE_TAB_STYLE = 'cursor: pointer; font-weight: normal; background: #eeeeee;'+
'min-width: 80px;';

// By default, chart tab (the map) is active
var activeTab = "chart";
var zoom = 1;

// Getting the content of the chart tab (the map)
// I put this content into a table. It has one row and two columns. The first column
// has another table inside it with only two rows and one column. First row is for zoom
// in and the second row is for zoom out. The second column of the main table has an
// iframe linking to map.py. I pass the zoom parameter (default value 1) to map.py.
// The iframe gives the ability to scrolling if necessary.
function getChartsTab() {
	return '<table cols="*,5"><tr>' +
		'		<td valign="top"><table cellspacing=2px width=20px><tr><td
style="background:'
		+'#eeeeee; cursor: pointer"><center><span
onclick="zoomIn()">+</span></td></tr>'+
		'<tr><td style="background: #eeeeee; cursor: pointer">'+
		'<center><span onclick="zoomOut()">-</span></td></tr></table></td>' +
		'<td><iframe style="border: solid gray 1px" width=570 height=570 '+
		'src="https://www.geos.ed.ac.uk/~s1757431/cgi-bin/map.py?zoom='+zoom+'"
/></td>'+
		'</tr></table>'  ;
}

// Getting the content of the fields table by linking to fields.py
function getFieldsTab() {

	return '<table><tr>'+
		'<td valign="top"> <iframe  width=750 height=750'+
		' src="https://www.geos.ed.ac.uk/~s1757431/cgi-bin/fields.py"/> </td>'+
		'</tr></table>'  ;

}

// Getting the content of the finds table by linking to finds.py
function getFindsTab() {

	return '<table><tr>'+
		'<td valign="top"> <iframe  width=750 height=750'+
```

```
                ' src="https://www.geos.ed.ac.uk/~s1757431/cgi-bin/finds.py"/> </td>'+
                '</tr></table>'   ;

}

// Zooming in (at most 10)
function zoomIn() {
        if (zoom < 10) {
                zoom++;
                updateTab();
        }
}

// Zooming out
function zoomOut() {
        if (zoom > 1) {
                zoom--;
                updateTab();
        }
}


//updating the content of the main part of the html based on the active tab.
function updateTab() {
        var content;
        if (activeTab == "chart") {
                content = getChartsTab();
                document.getElementById('chartTab').style = ACTIVE_TAB_STYLE;
                document.getElementById('fieldsTab').style = INACTIVE_TAB_STYLE;
                document.getElementById('findsTab').style = INACTIVE_TAB_STYLE;

        }

        else if (activeTab == "fields") {
                content = getFieldsTab();
                document.getElementById('chartTab').style = INACTIVE_TAB_STYLE;
                document.getElementById('fieldsTab').style = ACTIVE_TAB_STYLE;
                document.getElementById('findsTab').style = INACTIVE_TAB_STYLE;
        }

        else {
        content = getFindsTab();
                document.getElementById('chartTab').style = INACTIVE_TAB_STYLE;
                document.getElementById('fieldsTab').style = INACTIVE_TAB_STYLE;
                document.getElementById('findsTab').style = ACTIVE_TAB_STYLE;
        }
        document.getElementById('content').innerHTML = content;
}

// changing the activeTab to chart and calling updateTab
function showCharts() {
        activeTab = "chart";
        updateTab();
}
```

```
// changing the activeTab to fields and calling updateTab
function showFieldsTable() {
	activeTab = "fields";
	updateTab();
}

// changing the activeTab to finds and calling updateTab
function showFindsTables() {
	activeTab = "finds";
	updateTab();
}

</script>
</head>
<body style="background: #6a6a6b">
<center>

<!--
I put eveything insid a table. The first row has one column. Inside the column is another
smaller table with one row and three columns. Those columns are the different tabs.
The second row of the main table has only one common which will be later on filled with
the content of the selected tab.
 -->

<table style="border: none;">
	<tr>
		<td height="50px">
			<center>
				<table style="width: 120px; font-family: tahoma; font-size: 12px"
cellpadding=5px>
					<tr>
						<td id="chartTab" onclick="showCharts()"
align="center">Chart</td>
						<td id="fieldsTab" onclick="showFieldsTable()"
align="center">Fields Table</td>
						<td id="findsTab" onclick="showFindsTables()"
align="center">Finds Table</td>
					</tr>
				</table>
			</center>
		</td>
	</tr>
	<tr padding=20>
		<td id="content">

		</td>
	</tr>
</table>
</center>
</body>

<script>

// I first update based on the default tab. It will set the content of the "content" tab.
```

```
updateTab();

</script>
</html>
```

ui.js

```
// It will form a div with the information of one finding. It will be shown by clicking
// on a finding.
function getFindingInfoElement(e, ID,  x, y, type, depth, notes, W){
	var z = document.createElement('div');
	z.id = ID;
	var zx = e.pageX;
	var zy = e.pageY-100;
	width= 140;
	height= 100;
	if (zy<0){
		zy=zy+height;
	}
	if (zy+100>W){
		zy=zy-height;
	}
	if (zx<0){
		zx=zx+width;
	}
	if (zx+140>W){
		zx=zx-width;
	}
	z.style.cssText = "position: absolute; left: "+zx+"px; top:  "+zy+"px; opacity:;"
	+ "background: grey; width:" +width+"px; height:"+height+"px; border-radius: 5px; color:
black";
	z.innerHTML = "(X,Y,Z)= ("+x+","+y+","+depth+") <br> Type of Find:"+type+" <br> Field
notes:"+notes;
  return z;
}

// The above div will be shown by clicking the finding. If we click again, it will be
// removed by calling removeFindingInfo.
function showFindingInfo(e, ID,  x, y, type, depth, notes){
	var find_info = document.getElementById(ID);
	if (find_info){//I should remove because it has been pressed twice
		removeFindingInfo(e,ID);
	}
	else{//we should add
	find_info = getFindingInfoElement(e, ID,  x, y, type, depth, notes);
	document.getElementById('root').appendChild(find_info);
	}

}


// function showFindingInfo(e, ID,  x, y, type, depth, notes){
//	var find_info = document.getElementById(ID);
//	if (find_info || (Date.now()-lastAddTime)<2000){
```

```
//                      return;
//          }
//          lastAddTime = Date.now();
//          find_info = getFindingInfoElement(e, ID,  x, y, type, depth, notes);
//          document.getElementById('root').appendChild(find_info);
//          setTimeout(function(){ document.getElementById('root').removeChild(find_info) }, 5000);
// }

// Removes the info for the specified finding
function removeFindingInfo(e, ID){
            var find_info = document.getElementById(ID);
            if (find_info){
                        document.getElementById('root').removeChild(find_info)
            }
//          setTimeout(function(){var find_info = document.getElementById(ID);if
(find_info){document.getElementById('root').removeChild(find_info)}}, 2000);

}

//This function makes an AJAX call to delete one row from the specified table (tab).
// tab can be either "fields", or "finds".
function deleteTable(e,fid, tab) {
   var xhttp = new XMLHttpRequest();
   xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
                                    window.location.reload();
      }

   };

   var q = ''
   if (tab=='fields'){
            q = 'Are you sure you want to delete field "'+fid+'"?';
   }
   else{
              q = 'Are you sure you want to delete finding "'+fid+'"?';
   }

//    We ask the use if he/she is sure about deleting

   if (confirm(q)) {

      xhttp.open("POST", "https://www.geos.ed.ac.uk/~s1757431/cgi-bin/delete.py", true);
      xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
      xhttp.send("fid=" +fid+"&tab="+tab);
   }

}

// An AJAX call is done to reset the content of a table to its original value.
function resetTable(e, tab) {
            var xhttp = new XMLHttpRequest();
            xhttp.onreadystatechange = function() {
                        if (this.readyState == 4 && this.status == 200) {
                                    window.location.reload();
```

```
        }

        };


  xhttp.open("POST", "https://www.geos.ed.ac.uk/~s1757431/cgi-bin/reset.py", true);
  xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  xhttp.send("tab="+tab);
}

// Inserting a row into the fields table. The values are checked to be valid. After
// inserting, the window will be reloaded.
function insertField(e,LOWX,LOWY,HIX,HIY,OWNER,CROP) {
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
                if (this.readyState == 4 && this.status == 200) {
                        window.location.reload();
        }

        };

        if (!(LOWX && LOWY && HIX && HIY && OWNER && CROP)){
                alert("Please enter values for all the fields.");
                return -1;
        }

        LOWX = parseInt(LOWX);
        HIX = parseInt(HIX);
        LOWY = parseInt(LOWY);
        HIY = parseInt(HIY);

        if (LOWX<0 || LOWX>16 ){
                alert("LOWX must be between 0 to 16");
                return -1;
        }


        if (LOWY<0 || LOWY>16 ){
                alert("LOWY must be between 0 to 16");
                return -1;
        }

        if (LOWX>=HIX){
                alert("LOWX must be smaller than HIX");
                return -1
        }

        if (LOWY>=HIY){
                alert("LOWY must be smaller than HIY");
                return -1;
        }

        if (HIX<0 || HIX>16 ){
                alert("HIX must be between 0 to 16");
                return -1;
```

```
	}


	if (HIY<0 || HIY>16 ){
		alert("HIY must be between 0 to 16");
		return -1;
	}

	AREA = (HIX-LOWX)*(HIY-LOWY)/6.74;
	AREA = AREA.toFixed(2);


  xhttp.open("POST", "https://www.geos.ed.ac.uk/~s1757431/cgi-bin/insert.py", true);
  xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  post_msg =
"tab=fields&LOWX="+LOWX+"&LOWY="+LOWY+"&HIX="+HIX+"&HIY="+HIY+"&AREA="+AREA+"&
OWNER="+OWNER+"&CROP="+CROP;
  xhttp.send(post_msg);
}

// Inserting a row into the finds table. The values are checked to be valid. After
// inserting, the window will be reloaded.
function insertFind(e,XCOORD,YCOORD,TYPE,DEPTH,FIELD_NOTES) {
	var xhttp = new XMLHttpRequest();
	xhttp.onreadystatechange = function() {
		if (this.readyState == 4 && this.status == 200) {
			window.location.reload();
	}

	};

	if (!(XCOORD && YCOORD && TYPE && DEPTH && FIELD_NOTES)){
		alert("Please enter values for all the fields.");
		return -1;
	}

	XCOORD = parseInt(XCOORD);
	YCOORD = parseInt(YCOORD);

	if (XCOORD<0 || XCOORD>16 ){
		alert("XCOORD must be between 0 to 16");
		return -1;
	}


	if (YCOORD<0 || YCOORD>16 ){
		alert("YCOORD must be between 0 to 16");
		return -1;
	}

  xhttp.open("POST", "https://www.geos.ed.ac.uk/~s1757431/cgi-bin/insert.py", true);
  xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  post_msg =
"tab=finds&XCOORD="+XCOORD+"&YCOORD="+YCOORD+"&TYPE="+TYPE+"&DEPTH="+DEPTH+"
&FIELD_NOTES="+FIELD_NOTES;
```

```
    xhttp.send(post_msg);
}
```

map.py

```
#!/usr/bin/python3
import cgi
import cgitb
import cx_Oracle
import credentials

# This file contains the content of the 'Charts' tab.

my_user = credentials.login['user']
my  password = credentials.login['password']

MAX  Y = 16
MAX_X = 16

# Forms a tag for a SVG text.
# This function is responsible for creation of any text (string or number) in the grid tab.
# The inputs are X and Y positions, the text to be shown, amount of shift (x or y) and the font_size
# Using necessary transformations, we mirror the text so it will show well at the end!
def get_text_tagStr(X,Y,text,SHIFT_TEXT_X,SHIFT_TEXT_Y,FONT_SIZE=.3):
    Y = MAX_Y - Y
    ret = '<text x="'+str(X-SHIFT_TEXT_X)+'" y="'+str(Y+SHIFT_TEXT_Y)+'" font-family="Verdana" font-
size="'+\
          str(FONT  SIZE)+'" transform=" translate(0,'+str(MAX  Y)+') scale(1,-1)">'+str(text)+'</text>\n'
    return ret

# Forms a tag for a SVG rectangle
# This function creates rectangles with the use of 'rect' SVG tag. It draws fields and the container of their id
# numbers in the middle of each field.  get_text_tagStr is also called to fill the container with appropriate id number.
def get  field  tagStr(Field  Id,LOWX,LOWY,HIX,HIY):#TODO: move the attributes to a class
    ret = '<g>\n'
    ret += '<rect x="'+str(LOWX)+'" y="'+str(LOWY)+'" width="'+str(HIX-LOWX)+'" height="'+str(HIY-
LOWY)\
           +'" fill="none" stroke="black" stroke-width="0.05" />\n'

    CENTER_X = (LOWX+HIX)/2
    CENTER_Y = (LOWY+HIY)/2

    ret+= '<rect  x="'+str(CENTER_X-.3)+'" y="'+str(CENTER_Y-.3)+\
          '" width="0.6" height=".6" fill="none" stroke="black" stroke-width="0.05"/>'

    text_tag_str =
get_text_tagStr(CENTER_X,CENTER_Y,Field_Id,SHIFT_TEXT_X=.15,SHIFT_TEXT_Y=.15,FONT_SIZE=.4)
    ret += text_tag_str
    ret += '</g>\n'

    return ret


# It uses SVG circle tag to point the location of the finds on the grid tab.
# It also calls get  text  tagStr to allocate appropriate id number to each find.
def get_finding_tagStr(FIND_ID,XCOORD,YCOORD,TYPE,DEPTH,FIELD_NOTES,W):
    ret = '<circle style="cursor: pointer;" cx="'+str(XCOORD)+'" cy="'+str(YCOORD)+'" r=".15"
stroke="black"'+\
    ' stroke-width="0.05" fill="yellow" onclick="showFindingInfo(event,'+str(FIND_ID)+', '+str(XCOORD)+', '+\
    str(YCOORD)+', '+str(TYPE)+', '+str(DEPTH)+', \''+FIELD_NOTES+'\'' + ', ' + str(W)+')"/>'
```

```
    text_tag_str = get_text_tagStr(XCOORD,YCOORD,FIND_ID,SHIFT_TEXT_X=-
.2,SHIFT_TEXT_Y=0.15,FONT_SIZE=.4)
    ret += text_tag_str
    return ret

# It uses the rect and circle SVG tag to draw the legend for the map. Also calls get_text_tagStr to enter desired
#  texts to legend section.
def get_legend_tagStr():
    ret = '<g>\n'
    ret += '<rect x="'+str(12.8)+'" y="'+str(.8)+'" width="'+str(3)+'" height="'+str(3)+\
          '" stroke="black" stroke-width="0.05" fill="white" />\n'
    ret +='<circle " cx="'+str(13.5)+'" cy="'+str(1.5)+'" r=".15" stroke="black" stroke-width="0.05"
fill="yellow" />\n'
    ret +='<rect x="'+str(13.2)+'" y="'+str(2.2)+\
          '" width="0.6" height=".6" fill="none" stroke="black" stroke-width="0.05"/>'
    text_tag_str1 = get_text_tagStr(13.4,2.3,1,0,0,.4)
    text_tag_str2=get_text_tagStr(13.2,3.1,'Legend',0,0,.4)
    text_tag_str3=get_text_tagStr(14.2,2.3,'Field ID',0,0,.4)
    text_tag_str4=get_text_tagStr(14.2,1.3,'Find ID',0,0,.4)

    ret += text_tag_str1
    ret +=text_tag_str2
    ret +=text_tag_str3
    ret +=text_tag_str4

    ret += '</g>\n'
    return ret



cgitb.enable()
print("Content-Type: text/html\n")
print("<!DOCTYPE html>")

print("<head>")
print("<title>Web Mapping Project</title>")
print("</head>")

print("<body style='background: white'>")
print("<div id='root'>")
print ('<script type="text/javascript" src="../ui.js"></script>')

# Getting the value of the zoom. By default, zoom=1. The size of the original window (iframe) is fixed, but when we
# increase the zoom (always z<=10), we increase the width of the SVG (W). Therefore, it has the effect of zooming in.
arguments = cgi.FieldStorage()
W = int(arguments['zoom'].value) * 200 + 350


# Start the SVG
print('<svg width='+str(W)+' height='+str(W)+' viewBox= "-1 -1 18 18" style= "border: none;">')

print('<g transform="matrix(1,0,0,-1,0,16)">')

# Create the grid
for j in range(17):
    print (get_text_tagStr(-1+.4,j,str(j),0,0,.3))
    print (get_text_tagStr(j,-1+.4,str(j),0,0,.3))
    print('<path stroke="grey" stroke-width="0.025" d="M'+str(j)+' 0 v16"/>')
    print('<path stroke="grey" stroke-width="0.025" d="M0 '+str(j)+' H16"/>')

# Query the database to get the fields and findings
conn = cx_Oracle.connect(my_user+"/"+my_password+"@geosgen")
```

```
c = conn.cursor()
c.execute("select * from fields2")
for row in c:
   print(get  field  tagStr(row[0],row[1],row[2],row[3],row[4]))
conn.close()

conn = cx_Oracle.connect(my_user+"/"+my_password+"@geosgen")
c = conn.cursor()
c.execute("select * from finds2")
for row in c:
   s = row[5].lower()
   s = s[0].upper()+s[1:]
   print(get_finding_tagStr(row[0],row[1],row[2],row[3],row[4],s,W))
conn.close()

print(get_legend_tagStr ())

print('</svg>')
print("</div>")
print("</body>")
print("</html>")
```

fields.py

```
#!/usr/bin/python3
import cgi
import cgitb
import cx_Oracle
import credentials

my_user = credentials.login['user']
my_password = credentials.login['password']

# The get_field_row_table function makes one row of the html table, using for loop for every column in the fields table
def get_field_row_table(Field_Id,LOWX,LOWY,HIX,HIY,AREA,OWNER,CROP):
   all_cols = [Field_Id,LOWX,LOWY,HIX,HIY,AREA,OWNER,CROP]
   ret="<td id=\""+str(Field_Id)+"\" style=\"border:none; background-color: white; cursor: pointer\" " \
                        "onclick=\"deleteTable(event,"+str(Field_Id)+",'fields')\"> &#x2716 </td>"
   for val in all_cols:
      column='<td>'+str(val)+'</td>'
      ret+= column

   ret2= '<tr>'+ret+'</tr>'
   return ret2

# This function creates the tag for inserting a new row (field). It returns a row which will be added to the main table.
# The types of the fields are taken into account
def get_insert_row():
   ret = "<td class='invis'><input size='8' type='submit' value='Add Field'>  </td><td class='invis'></td>"

   names = 'LOWX,LOWY,HIX,HIY,OWNER'.split(',')
   types = ['number','number','number','number','text']


   for i in range(4):
      column= '<td class="invis"><input style="width: 100%" type="'+types[i]+'" name="'+names[i]+'" id="'\
            +names[i]+'" placeholder="'+names[i]+'"></td>'
      ret+= column
```

```
    ret+="<td class='invis'></td>"

    i=4
    column= '<td class="invis"><input style="width: 100%" type="'+types[i]+'" name="'+names[i]+\
            '" id="'+names[i]+'" placeholder="'+names[i]+'"></td>'
    ret+= column


    select_str = \
        '<select name="CROP" id="CROP"><option value="1">TURNIPS</option><option value="2">OIL
SEED RAPE</option>'+\
                '<option value="3">STRAWBERRIES</option>' \
                '<option value="4">PEAS</option><option value="5">POTATOES</option></select>'



    column= '<td class="invis">'+select_str+'</td>'
    ret+=column

    ret2= '<tr>'+ret+'</tr>'
    ret2 ='<form action="' \

'javascript:insertField(event,LOWX.value,LOWY.value,HIX.value,HIY.value,OWNER.value,CROP.value)">'\
          +ret2+'</form>'
    return ret2

cgitb.enable()
print("Content-Type: text/html\n")

print("<!DOCTYPE html>")

print('<head>')

print('<style>')
print('table {font-family: arial, sans-serif; border-collapse:collapse; width: 100%; text-align: center;}')
print(' th,td {border: 1px solid black; text-align: left; padding: 8px;}')
print('table tr:nth-child(even) {background-color: #eee;}')
print('table tr:nth-child(odd) {background-color:#fff;}')
print('table th {background-color: black;color: white;}')
print('td.invis { border:none; background-color: white; border-collapse:collapse}')
print('</style>')

print('</head>')

print("<body style='background: white'>")
print('<center>')
print ('<script type="text/javascript" src="../ui.js"></script>')
# The main table
print('<table id="field_table">')
# Head of the table
print('<tr>')
print('<th style=\"border:none; background-color: white;\"></th>')
print('<th>FIELD ID</th>')
print('<th>LOWX</th>')
print('<th>LOWY</th>')
print('<th>HIX</th>')
print('<th>HIY</th>')
print('<th>AREA</th>')
print('<th>OWNER</th>')
print('<th>CROP</th>')
print('</tr>')
```

```
conn = cx_Oracle.connect(my_user+"/"+my_password+"@geosgen")
c = conn.cursor()

# Querying the fields table to get the fields (also joining with crops to get the name of the crops)

c.execute("select
fields2.Field_Id,fields2.LOWX,fields2.LOWY,fields2.HIX,fields2.HIY,fields2.AREA,fields2.OWNER"+
        ",gisteach.crops.name from fields2, gisteach.crops where"+
        " fields2.crop = gisteach.crops.crop order by fields2.field_id")

# Adding all the rows the html table
for row in c:
    area = row[5]
    area = round (float (area),2)
    print(get_field_row_table(row[0],row[1],row[2],row[3],row[4],area,row[6],row[7]))

# Adding the insert portion
print(get_insert_row())

conn.commit()
conn.close()

print('</table>')
print('<br>')
print('<div>')
print("<button type=\"button\" onclick=\"resetTable(event,'fields')\">RESET</button>")
print('</div>')
print('</body>')
print('</center>')

print('</html>')
```

finds.py

```
#!/usr/bin/python3
import cgi
import cgitb
import cx_Oracle
import credentials

my_user = credentials.login['user']
my_password = credentials.login['password']

# The get_find_row_table function makes one row of the table, using for loop for every column in the finds table
def get find row table(FIND ID,XCOORD,YCOORD,TYPE,DEPTH,FIELD NOTES):
    all_cols = [FIND_ID,XCOORD,YCOORD,TYPE,DEPTH,FIELD_NOTES]
    ret="<td style=\"border:none; background-color: white; cursor:
pointer\"onclick=\"deleteTable(event,"+str(FIND_ID)+\
        ",'finds')\"> &#x2716 </td>"
    for val in all_cols:
        column='<td>'+str(val)+'</td>'
        ret+= column

    ret2= '<tr>'+ret+'</tr>'
    return ret2

# This function creates the tag for inserting a new row (field). It returns a row which will be added to the main table.
# The types of the fields are taken into account
def get_insert_row():
    ret = "<td class='invis'><input size='8' type='submit' value='Add Find'>  </td><td class='invis'></td>"
```

```
    names = 'XCOORD,YCOORD,TYPE,DEPTH,FIELD_NOTES'.split(',')
    types = ['number','number','xxx','xxx','text']

    # XCOORD, YCOORD
    for i in range(2):
        column= '<td class="invis"><input style="width: 100%" type="'+types[i]+'" name="'+names[i]+'"
id="'+names[i]+\
                '" placeholder="'+names[i]+'"></td>'
        ret+= column

    i+=1
    # TYPE
    select_str = '<select name="TYPE" id="TYPE"><option value="1">SHARD</option><option
value="2">METAL_WORK</option>'+\
                 '<option value="3">FLINT</option><option value="4">BONE</option></select>'

    column= '<td class="invis">'+select_str+'</td>'
    ret += column

    # DEPTH
    i+=1

    column= '<td class="invis"><input style="width: 100%" type="number" min="0" step=".01"
name="'+names[i]+'" id="'+\
            names[i]+'" placeholder="'+names[i]+'"></td>'
    ret+= column

    # FIELD_NOTES
    i+=1

    column= '<td class="invis"><input style="width: 100%" type="text"  name="'+names[i]+'" id="'+names[i]+\
            '" placeholder="'+names[i]+'"></td>'
    ret+= column

    ret2= '<tr>'+ret+'</tr>'
    ret2 ='<form action="javascript:insertFind(' \

'event,XCOORD.value,YCOORD.value,TYPE.value,DEPTH.value,FIELD_NOTES.value)">'+ret2+'</form>'
    return ret2

cgitb.enable()
print("Content-Type: text/html\n")

print("<!DOCTYPE html>")

print('<head>')
print('<style>')
print('table {font-family: arial, sans-serif; border-collapse:collapse; width: 100%; text-align: center;}')
print(' th, td {border: 1px solid black; text-align: left; padding: 8px;}')
print('table tr:nth-child(even) {background-color: #eee;}')
print('table tr:nth-child(odd) {background-color:#fff;}')
print('table th {background-color: black;color: white; font-size:13px}')
print('td.invis { border:none; background-color: white; border-collapse:collapse}')



print('</style>')
print('</head>')
print("<body style='background: white'>")
print('<center>')
print ('<script type="text/javascript" src="../ui.js"></script>')
# The main table
```

```
print('<table id = "finds_table"')
# Head of the table
print('<tr>')
print('<th style=\"border:none; background-color: white\"> </th>')
print('<th> FIND ID</th>')
print('<th>XCOORD</th>')
print('<th>YCOORD</th>')
print('<th>TYPE</th>')
print('<th>DEPTH</th>')
print('<th>FIELD NOTES</th>')
print('</tr>')

conn = cx_Oracle.connect(my_user+"/"+my_password+"@geosgen")
c = conn.cursor()

# Querying the finds table to get the find (also joining with class to get the type of the findings)

c.execute("select finds2.FIND_ID,finds2.XCOORD,finds2.YCOORD, gisteach.class.NAME,finds2.DEPTH,
finds2.FIELD_NOTES"
        " from finds2, gisteach.class"+ " where finds2.type = gisteach.class.type order by finds2.FIND_ID")

# Adding all the rows the html table
for row in c:

   print(get_find_row_table(row[0],row[1],row[2],row[3],row[4],row [5]))

# Adding the insert portion
print (get_insert_row())

conn.commit()
conn.close()


print('</table>')
print('<br>')
print('<div>')
print("<button type=\"button\" onclick=\"resetTable(event,'finds')\">RESET</button>")
print('</div>')

print('</center>')

print('</body>')
print('</html>')
```

reset.py

```
#!/usr/bin/python3
import cx_Oracle
import cgi
import cgitb
import credentials

my_user = credentials.login['user']
my_password = credentials.login['password']

cgitb.enable()
print("Content-Type: application\n")
conn = cx_Oracle.connect(my_user+"/"+my_password+"@geosgen")
```

```
c = conn.cursor()

arguments = cgi.FieldStorage()
tab = arguments['tab'].value

# We reset by reading from the original tables gisteach.fields or gisteach.finds

if tab=="fields":
   c.execute("delete from fields2 where 1=1")
   c.execute("insert into fields2 (select * from gisteach.fields)")

elif tab=="finds":
   c.execute("delete from finds2 where 1=1")
   c.execute("insert into finds2 (select * from gisteach.finds)")

# c.execute("insert into fields2 (select * from gisteach.fields)")

conn.commit()
conn.close()
```

delete.py

```
#!/usr/bin/python3
import cx_Oracle
import cgi
import cgitb
import credentials

my_user = credentials.login['user']
my_password = credentials.login['password']

cgitb.enable()
print("Content-Type: application\n")
conn = cx_Oracle.connect(my_user+"/"+my_password+"@geosgen")
c = conn.cursor()

# Retrieving the table and id arguments
arguments = cgi.FieldStorage()
fid = arguments['fid'].value
tab = arguments['tab'].value

# Deleting from the table
if tab=="fields":
   c.execute("delete from fields2 where field_id="+str(fid))
elif tab=="finds":
   c.execute("delete from finds2 where find_id="+str(fid))

conn.commit()
conn.close()
```

insert.py

```
#!/usr/bin/python3
import cx_Oracle
import cgi
import cgitb
import credentials
```

```
my_user = credentials.login['user']
my  password = credentials.login['password']

cgitb.enable()
print("Content-Type: application\n")
conn = cx_Oracle.connect(my_user+"/"+my_password+"@geosgen")
c = conn.cursor()

arguments = cgi.FieldStorage()


# Retrieving the table name (fields or finds)
tab = arguments['tab'].value

# The argument keys we're looking for based on the table

if tab=='fields':
   keys = 'LOWX,LOWY,HIX,HIY,AREA,OWNER,CROP'.split(",")
else:
   keys = 'XCOORD,YCOORD,TYPE,DEPTH,FIELD_NOTES'.split(",")

# Retrieving the values of columns
values = []
for k in keys:
   print ("k: ",k)
   values.append(str(arguments[k].value))

# Getting the max_id of the table
if tab=='fields':
   rows = c.execute("select max(field_id) from fields2")
elif tab=='finds':
   rows = c.execute("select max(find_id) from finds2")

for row in rows:
   max_id = row[0]

# new_id is max_id+1
max_id += 1

# Inserting into the table
if tab=='fields':
   c.execute("INSERT INTO fields2 VALUES ("+ str(max  id)+","+  values[0]+","+ values[1]+","+
values[2]+","+
         values[3]+","+ values[4]+",'"+ values[5]+"'," + values[6]+ ")")
elif tab=='finds':
   c.execute("INSERT INTO finds2 VALUES ("+ str(max_id)+","+  values[0]+","+
         values[1]+",'"+ values[2]+"',"+ values[3]+",'"+ values[4]+"')")

conn.commit()
conn.close()
```