CHAPTER 5

KSR-QUADTREE: AN INTELLIGENT KNOWLEDGE STORAGE AND RETRIEVAL USING QUADTREE STRUCTURE

5.1 INTRODUCTION

Data volume with respect to modern systems has been growing outwardly at a rapid rate. The systems also face a tough task of frequent re-scanning of the large datasets stored because of the updation process. Frequent re-scanning, updation and large corpus data leads to having larger retrieval time and decreased efficiency. Hence, the data retrieved by automated systems may not be efficient and accurate. Knowledge Based System using Quad tree is employed in this chapter. The Quad trees designed in the chapter helps in retrieving knowledge about the suitable plants for the given kind of soil. The Quad tree is built using the knowledge and information given by Edaphologists and domain experts. The system is made of two modules, namely Quad Tree construction module and information retrieval module. In the Quad tree module, obtained data is transformed to a dynamic Quad tree based knowledge data structure. In the information retrieval module, plant names are retrieved based on the input soil characteristics by the user. In the retrieval process, fuzzy functions are employed. The system is implemented using .NET/SQL. The obtained results prove that the system has performed well by having good evaluation metric values.

5.2 AN INTELLIGENT KNOWLEDGE STORAGE AND RETRIEVAL USING QUADTREE STRUCTURE

The system consists of two modules of Quad Tree construction module and information retrieval module. In the Quad Tree construction, Quad tree is
built from the data given by Edaphologists. In the first module, the records from the database are converted into 2D points which are then stored recursively in a two-dimensional space with four recursive quadrants. In the information retrieval module, knowledge retrieval is carried out with the help of constructed knowledge base (XML). The block diagram of the system is given in Figure 5.1.

![Figure 5.1 Block diagram of the KSR-Quadtree](image)

### 5.2.1 Quad Tree Terminology

A Quad tree is a type of tree data structure where each internal node has precisely four children. Quad trees are regularly employed in dividing a two-dimensional space into four quadrants recursively. It may also divide into other shapes such as rectangle. Quad trees possess the property of partitioning the space into flexible cells and each cell has a defined maximum capacity. There are many types of Quad trees based on the type of data they handle and in this system the point quad tree is employed.
The point Quad tree is a binary tree version which is engaged to symbolize two dimensional point data. Point Quad tree inherits all the properties of Quad trees but varies from others in the fact that center of subdivision is always on a point which makes it a true tree. The shape of the tree is based on the order in which the data is executed. The nodule of the point Quad tree is like the nodule of a binary tree except the fact that it has four pointers in its place of two for binary tree. The four pointers in the Quad tree are named as top-right, top-left, bottom-right and bottom-left. In case of binary pointers being named left and right. Figure 5.2 gives a model of Quad tree. The (x, y) coordinates of the Quad tree is termed as key. Hence the nodule comprises of:

- Pointers (top-right, top-left, bottom-right, bottom-left)
- Where, each pointer has a key ((x, y) coordinates)
- The value (which is the side of the quadrant)

![Figure 5.2 Model of Quad tree structure](image)

There are some properties that Quad tree possesses and they are given below:
• The Quad tree partitions the space into adaptable cells.
• Every cell present in the Quad tree has a defined capacity. Suppose the maximum capacity is exceeded, the cell is divided further.
• The Quad tree pursues the spatial decomposition of features from the database.
• Quad tree data is generally precise as it performs multiple divisions to arrive at the exact data.

In this system, Quad tree points are transformed to the form of XML tags as it reduces the retrieval time and results in having good retrieval and the fact that it employs nearest neighboring search.

5.2.2 2-D Quad Tree based Knowledge Design

In a Quad tree based system, rescanning of the database can be evaded; hence disk access and the retrieval time can be minimized. It can also decrease the space complexity as the large corpus data is converted into a file format which can be stored inside the main memory. This type of systems can co-ordinate with humans to have quality systems. This has the benefits in the areas of presentation, documentation of knowledge, intelligent decision support, self learning, reasoning and explanation. The system is intelligent in knowledge storage and retrieval using Quad tree structure which is designed for having the right kind of plants for the user input soil features. Quad tree construction module and information retrieval module are the two modules in the system. Having the right kind of plant for the given soil conditions is very essential and can lead to better yield. That is by knowing the soil features suitable for the plant, the chance of having higher output increases. Soil features play a major role in the productivity of the plants. Planting a crop in unsuitable soil features may not yield any profit and would bring loss to the farmer. Hence, the knowledge of plants suited for the given soil conditions are of paramount importance. This system aims to achieve this goal of having the right plant for
the given soil conditions. Hence, this system would help and assist Edaphologists and agriculturists in a big way.

The information or the dataset is obtained from various Edaphologists and those related with plants and soils. These plants and the related soil features form the dataset. It mainly consists of attributes like plant name, plant identification number, geology, vegetation type and also content of various elements (like calcium, sodium, clay, sand, potassium, electrical conductivity etc.) of the soil. The information is stored in the XML Quad tree form after a series of processes. Initially, data preparation is carried out with the use of type conversion and normalization. It is then reduced to two dimensional databases which are then transformed to 2D Quad tree. From this, XML based Quad tree is created. Block diagram of the Quad tree construction module where the input data from Edaphologists are converted to XML based Quad tree is given in Figure 5.3. An example based on the soil and plant information collected from the Edaphologists is given in Table 5.1.

![Diagram of the Quad tree construction module](image-url)

**Figure 5.3 Block diagram of the Quad tree construction module**
### Table 5.1 Site characteristics and Plant Names- A Sample

<table>
<thead>
<tr>
<th>Site Characteristics &amp; Plant names</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pedon Number</strong>: 0419</td>
</tr>
<tr>
<td><strong>Series name</strong>: Valvangi</td>
</tr>
<tr>
<td><strong>Location</strong>: Villatikulam</td>
</tr>
<tr>
<td><strong>Latitude</strong>: 9°15’52.92”</td>
</tr>
<tr>
<td><strong>Longitude</strong>: 78°15’44.28”</td>
</tr>
<tr>
<td><strong>Elevation</strong>: 35m above MSL</td>
</tr>
<tr>
<td><strong>Slope</strong>: Very gently sloping</td>
</tr>
<tr>
<td><strong>Erosion</strong>: slight erosion</td>
</tr>
<tr>
<td><strong>Drainage</strong>: Imperfectly drained</td>
</tr>
<tr>
<td><strong>Permeability</strong>: Very slow</td>
</tr>
<tr>
<td><strong>Geology</strong>: Clay</td>
</tr>
<tr>
<td><strong>Vegetation</strong>: Neem,prosopis Julifera</td>
</tr>
<tr>
<td><strong>Classification</strong>: Clayey,isohyperthermic, mixed,calcareous,lithic Haplustepts</td>
</tr>
</tbody>
</table>

Pedon Description of the soil is given in Table 5.2 which shows the depth and the respective soil descriptions. This is converted into parent plant database and child description database. Plant database consists of plant details and is given in Figure 5.4. It contains the details about the plant such as plant name, plant identification number, geology and taxonomy. Description table consist of the details about the soil and is given in Figure 5.5. Soil features include depth, clay, silt, sand, Ph, electrical conductivity, Calcium, Magnesium, Sodium, Potassium and Phosphorus Pent-oxide, Potassium Oxide values. The table also consists of plant identification number which acts as a foreign key to join the tables.
Table 5.2 Soil Description

<table>
<thead>
<tr>
<th>Depth</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-11</td>
<td>Dark gray, clay, weak, medium, few roots, kankar modules, clear, smooth boundary</td>
</tr>
<tr>
<td>11-27</td>
<td>Very dark gray, clay, moderate, fine, few pores, very fine, violent effervescence, wavu boundary</td>
</tr>
<tr>
<td>27+</td>
<td>Kankar mixed granitic modules</td>
</tr>
</tbody>
</table>

Figure 5.4 Sample parent Plant database

Figure 5.5 Sample child Description database
5.2.3 Data Preparation

Data preparation is the first step for converting the information obtained from Edaphologists to XML Quad tree format. It involves two steps of type conversion and normalization. Here, initially the mean values of the attributes are computed and subsequently type converted and normalized.

5.2.3.1 Type Conversion

The description table consists of 17 soil features in which some are in the alphanumerical format. When the input is in the form of alphanumeric value, the system would not be able to access it; hence the value is converted to numeric values. The soil features are transformed to unsigned integer format. The feature values indicate the content or state of the feature in the soil samples taken. Here, the feature values are given with the separation of comma. An illustration example is given as:

[Soil Color, Soil type, Soil Strength, Moisture, Coarseness]

Categorical variables like soil color, strength and moisture are given in string format of multiple words like “light brownish grey”. The categorical value is separated into a single word and every word is allocated a single byte. Different categorical values for the different soil features are given in Table 5.3.

**Table 5.3 Categorical values for the soil features**

<table>
<thead>
<tr>
<th>Soil Feature</th>
<th>Categorical Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soil Color</td>
<td>[Brown, Gray, Red, White, Black]</td>
</tr>
<tr>
<td>Soil Type</td>
<td>[Sand, Clay, Sandy Clay, Clay Loam, sandy Clay Loam]</td>
</tr>
<tr>
<td>Soil Strength</td>
<td>[Weak, Medium, Moderate, Strong, Structure less]</td>
</tr>
<tr>
<td>Soil Moisture</td>
<td>[Sticky, Slightly Sticky, very Sticky, Non-sticky]</td>
</tr>
<tr>
<td>Soil Coarseness</td>
<td>[Fine, single grained, fine granular]</td>
</tr>
</tbody>
</table>
For every categorical variable, a lookup table is employed to substitute categorical values with numeric values. The conversion of the categorical data to numeric value is carried out as mentioned in the examples.

**Example 1: “Light Brownish Grey”**

MSB – Byte (“Light”):

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

(“Brownish”):

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

LSB – Byte (“Grey”):

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Hence, Light brownish grey = $2^{16} + 2^8 + 2^1$

**Example 2: “Light Yellowish Brown”**

MSB – Byte (“Light”)

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

(“Yellowish”)

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

LSB – Byte (“Brown”)

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

Hence, Light yellowish brown = $2^{16} + 2^9 + 2^2$

Finally, the categorical values are transformed into numeric values in the respective columns and a 17-column vector each with numerical values is
obtained.

5.2.3.2 Normalization

After type conversion, normalization is carried out to confine the values of diverse ranges to a defined specific range. Here, the values from all ranges are converted to a defined range of zero to one. After normalization, 7 column vectors (for 17 features of plant and soil) where each column value ranges between 0 and 1 are obtained. The normalization technique employed in this system is the min-max normalization technique which is used to have the feature values in the range of 0 to 1. The Pseudo code of the min-max normalization technique is given below:

```plaintext
For i = 1: Na

Min_{old} = Find Minimum (i)

Max_{old} = Find Minimum (i)

Ni = \left( \frac{i - Min_{old}}{Max_{old} - Min_{old}} \right) \times \left( Max_{new} - Min_{new} \right) + Min_{new}

Where, Max_{new} - specified maximum value to be converted

Min_{new} - specified minimum value to be converted

Max_{old} - Original max value of the variable

Min_{old} - Original min value of the variable

Ni = Normalized values.
```
5.2.4 Conversion of Multidimensional Database into 2D Database

Multidimensional (17 dimensional) values are subsequently transformed to two-dimensional values of length and angle.

**Length Vector:** The length vector value \( L_{\text{vec}} \) can be computed by the formula:

\[
L_{\text{vec}} = \sqrt{C_1^2 + C_2^2 + \ldots + C_{17}^2}
\]  
(5.1)

Where, \( C_i \) is the value of \( i^{th} \) column and \( 1 \leq i \leq 17 \). The length vector is found out as the root of squared sum of the columns. In order to calculate vector angle certain additional quantities need to be calculated.

**Vector length (pts):** The vector length is calculated with the help of length vector. The pseudo-code is given by:

```
For g = 1; G = 1
L_{\text{vec}} = L_{\text{vec}} \cdot \text{pow}(pts[g],2)
End For
```

**Reference Points (pts, size):** For every data, the reference points are found out. The pseudo-code for finding the reference points \( Rfp \) are given by:

```
Rfp = []
For g = 1; G = 1
Rfp(g) = \left(\text{Floor} \left(\frac{pts[g]}{size}\right) \times size\right) + \left(\frac{size}{2}\right)
End For
```
**Vector Product (pts, Rfp):** The vector product is computed using the product of reference points and the values in the columns. The pseudo-code is given by:

```
For g = 1: G - 1
   Vp = Vp \\ (pts[g]* Rfp[g])
End For
```

**Vector Angle (pts, rfpts):** The vector angle is computed by using vector length of pts, vector length (reference points), vector product between pts and reference points. The pseudo-code is given by:

```
Ple = Vectorlength(pts)
Rple = Vectorlength(rfpts)
Vp = Vectorproduct(pts, rfpts)
Angle = Vp / ((Ple * Rple)
```

The conversion to 2 Dimension is given in the pseudo-code below:

```
Size = 2, Th = 0.001;
d = 0, L = 0, angle = 0;
L = Vectorlength(pts)
Do
   Size = Size / 2;
   Rfp = Referencepoint(pts, size)
   d = Eq dist angle(pts, Rfp)
   angle = angle + vectorangle(pts, Rfp)
While(d > th);
2-dimensional points are given by:
V2d = [L, angle]
```
5.2.5 **Construction of 2D Quad Tree**

After the conversion to 2-dimension, Quad tree is built from this data with the help of points. The input to the Quad tree will be length, angle and the plant identification number. The various steps involved in the Quad tree construction are given below:

A. Initially, the graph area is partitioned into four quadrants of Top Left, Top Right, Bottom Left and Bottom Right based on the range.

B. The opening set of input values are plotted in any one of the related quadrant.

C. Subsequently, for further input values, the plot is made after computing the deviation compared to the previously plotted value.

a. The condition is checked that if the threshold is greater than the size of the quadrant and if it is satisfied, the repeated splitting of the Quad tree is stopped. Subsequently, the x-side and y-side of the region is computed.

b. For every node, the related x, y coordinates and side values are found. The region value is computed from the formula: X side of the region = x-coordinate + side/2 and Y side of the region = y-coordinate + side/2

D. The step C is repeated till all the input values are plotted in the Quad tree.

E. Subsequently, the Quad tree is constructed for all points with root node as Null, the points in the main quadrants as the child of root node and the point in the sub quadrants as the child of main quadrants.

The pseudo code for the Quad tree construction is given below. In the code some representations are made and are defined as:

*pta* − Point which is to be added into the Quad tree.

*noid* − The node identification number.

*dis* − The distance between node’s coordinates and point’s coordinates.
th - It represents the threshold value set to add a point to the node.

```
flag = 0;

Function addPo int Quadtree (Po int Pta)

While(!flag)

Noid = searchnode(pt, Noid)

Dis = distance(pt, Noid)

if (dis > Th)

    Splitnode(Noid)

Else

    Addpo int node(pt, Noid)

Flag = 1;

End If

EndWhile
```

**Figure 5.6 The pseudo code of Construction of 2D Quadtree**

### 5.2.6 Construction of XML Quad Tree

The constructed 2D Quad tree is applied with a function to have the XML quadtree. XML Quad tree has the advantage of having better and faster retrieval compared to normal Quad tree. Once the conversion to the XML Quad tree is completed, the tree is stored in the file. The steps involved in converting to the XML Quad tree are given below:

A. The fields required like top right, top left, bottom left, bottom right for XML Quad tree are précised and specified.

B. The root node of the XML Quad tree is taken as zero.
C. The node identification number of the Quad tree is given to the corresponding root identification number of XML Quad tree.

D. The points in the least divided area are employed to map the plant information from the plant database.

E. Lastly, XML Quad tree is built with already navigated region in Quad tree with the plant details in the least divided region.

```plaintext
For N = 1 to M (quadtree count)
    node = quadtree[N]
    Elem = XMLCreateElement(node.name)
    If(node.haspoints)
        Point spts[] = node.getpo int s
        For j = 1 to pts.count
            Pec = XMLCreateElements(italic plant)
            Pec.addstring(pts[j], plantname)
            Elem.add(pec)
            Pec2 = XMLCreateElements(italic plant)
            Pec2.addstring(pts[j], geo log y)
            Elem.add(pce2)
            Pec3 = XMLCreateElements(italic taxonomy)
            Pec3.addstring(pts[j], taxonomy)
            Elem.add(pce3)
        End for
    End If
End for
```

Figure 5.7 The pseudo code of construction of XML Quadtree
5.2.7 Knowledge Retrieval

In this system, effective knowledge retrieval is carried out with the aid of XML architecture and XML tags. The system is designed for retrieving best plants for the input soil condition given by the user. When the user gives a query with input features, the suited plants are retrieved from the XML Quad tree. The information given by our system would benefit Edaphologists in knowing the right plants to be grown for the input soil conditions. That is right plant for the right soil characteristics are obtained with the aid of the system. The pseudo-codes for knowledge retrieval from XML and XML search are given below:

```
Function Search xml_element (pt[], mi)
    mx[4,2], Element Ele[4];
    Element EI = XML.GetElementByID(mi)
    While(EI.GetAttribute(ital IS PARENT) = 1) node
        Ele[1] = EI1.NextChildElement(ital TopLeft)
        mx[1:..] = EI1.GetAttributes(A, B)
        mx[2:..] = EI2.GetAttributes(A, B)
        mx[3:..] = EI3.GetAttributes(A, B)
        mx[4:..] = EI4.GetAttributes(A, B)
        id = findneares t _ node_u (pt, mx)
        mi = Ele[id].GetAttribute(ital NID)
        EI = Xml.GetElementByID(mi)
    End While
    Return mi
End function
```

Figure 5.8 The pseudo code of Knowledge retrieval from XML
Function Add plant xml (pt[], plant)

mx[1,2];

mi = 0;

Flag = False;

While(!flag)

mi = Search xml element (pt, mi)

Element El = XML.GetElementById(mi)

dis = distance(pt, mx)

if (dis > Th)

Split XML node (El)

Else

Add plant XML nodes (El, plant)

Flag = True;

End If

End While

End Function

Figure 5.9 The pseudo code of XML Search

Hence, information is retrieved effectively from the system and the system would benefit the Edaphologists and agriculturists in a great way, increasing the productivity by making them know the suitable plants for the given soil conditions.
5.3 EXPERIMENTAL RESULTS

This section presents the experimental results of the Quad tree-based knowledge retrieval system. The system is implemented in .NET\SQL.

5.3.1 Screen Shots, Sample query sets and output

The screenshot of the system is given in Figure 5.6. A sample query and its corresponding output is given in Table 5.4. A query set is generated by combining six queries to evaluate the performance of the system. The sample query set is given in Table 5.5 and the number of plants retrieved correspondingly is given in Table 5.6.

![Screen Shot of the system](image-url)

Figure 5.10 Screenshot of the system
Table 5.4 Sample query and its output

<table>
<thead>
<tr>
<th>Input Query</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Depth:</strong> 0-18</td>
<td><strong>Name:</strong> Prosopis juliflora</td>
</tr>
<tr>
<td><strong>Color:</strong> red</td>
<td><strong>Geology:</strong> Granite</td>
</tr>
<tr>
<td><strong>Sand:</strong> sandy clay loam</td>
<td><strong>Taxonomy:</strong> Clayey, mixed, isohyperthermic, noncalcareous, Typic Rhodustalfs</td>
</tr>
<tr>
<td><strong>Strength:</strong> moderate medium subangular blocky</td>
<td></td>
</tr>
<tr>
<td><strong>Moist:</strong> slightly sticky</td>
<td></td>
</tr>
<tr>
<td><strong>Pores:</strong> common pores</td>
<td></td>
</tr>
<tr>
<td><strong>Clay:</strong> 24.60</td>
<td></td>
</tr>
<tr>
<td><strong>Silt:</strong> 15.2</td>
<td></td>
</tr>
<tr>
<td><strong>Sand:</strong> 60.2</td>
<td></td>
</tr>
<tr>
<td><strong>PH:</strong> 7.36</td>
<td></td>
</tr>
<tr>
<td><strong>EC:</strong> 0.04</td>
<td></td>
</tr>
<tr>
<td><strong>CA:</strong> 8.0</td>
<td></td>
</tr>
<tr>
<td><strong>Mg:</strong> 1.5</td>
<td></td>
</tr>
<tr>
<td><strong>Na:</strong> 0.68</td>
<td></td>
</tr>
<tr>
<td><strong>K:</strong> 0.32</td>
<td></td>
</tr>
<tr>
<td><strong>P2O5:</strong> 29.45</td>
<td></td>
</tr>
<tr>
<td><strong>K2O:</strong> 228.00</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.5 Input Query set

<table>
<thead>
<tr>
<th>Query 1</th>
<th>Query 2</th>
<th>Query 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Depth</strong> = 26-52</td>
<td><strong>Depth</strong> = 50-91</td>
<td><strong>Depth</strong> = 13-33</td>
</tr>
<tr>
<td><strong>Color</strong> = light grey to grey</td>
<td><strong>Color</strong> = brownish yellow</td>
<td><strong>Color</strong> = reddish brown</td>
</tr>
<tr>
<td><strong>Sand</strong> = sandy clay</td>
<td><strong>Sand</strong> = clay</td>
<td><strong>Sand</strong> = sandy clay</td>
</tr>
<tr>
<td><strong>Strength</strong> = medium moderate subangular blocky</td>
<td><strong>Strength</strong> = medium moderate subangular blocky</td>
<td><strong>Strength</strong> = medium weak subangular blocky</td>
</tr>
<tr>
<td><strong>Moist</strong> = sticky</td>
<td><strong>Moist</strong> = sticky</td>
<td><strong>Moist</strong> = sticky</td>
</tr>
<tr>
<td><strong>Pores</strong> = pores</td>
<td><strong>Pores</strong> = few pores</td>
<td><strong>Pores</strong> = few pores</td>
</tr>
<tr>
<td><strong>Clay</strong> = 36.31</td>
<td><strong>Clay</strong> = 38.89</td>
<td><strong>Clay</strong> = 47.00</td>
</tr>
<tr>
<td><strong>Silt</strong> = 13.51</td>
<td><strong>Silt</strong> = 20.25</td>
<td><strong>Silt</strong> = 22.00</td>
</tr>
<tr>
<td><strong>Sand</strong> = 50.18</td>
<td><strong>Sand</strong> = 50.18</td>
<td><strong>Sand</strong> = 8.00</td>
</tr>
<tr>
<td><strong>PH</strong> = 6.28</td>
<td><strong>PH</strong> = 8.65</td>
<td><strong>PH</strong> = 8.00</td>
</tr>
<tr>
<td><strong>EC</strong> = 9.40</td>
<td><strong>EC</strong> = 0.09</td>
<td><strong>EC</strong> = 1.50</td>
</tr>
<tr>
<td><strong>CA</strong> = 9.40</td>
<td><strong>CA</strong> = 15.34</td>
<td><strong>CA</strong> = 14.00</td>
</tr>
<tr>
<td><strong>Mg</strong> = 1.80</td>
<td><strong>Mg</strong> = 7.82</td>
<td><strong>Mg</strong> = 10.00</td>
</tr>
<tr>
<td><strong>Na</strong> = 0.50</td>
<td><strong>Na</strong> = 3.01</td>
<td><strong>Na</strong> = 1.10</td>
</tr>
<tr>
<td><strong>K</strong> = 0.12</td>
<td><strong>K</strong> = 0.12</td>
<td><strong>K</strong> = 0.60</td>
</tr>
<tr>
<td><strong>P2O5</strong> = 11.26</td>
<td><strong>P2O5</strong> = 14.00</td>
<td><strong>P2O5</strong> = 7.00</td>
</tr>
<tr>
<td><strong>K2O</strong> = 108.00</td>
<td><strong>K2O</strong> = 302.00</td>
<td><strong>K2O</strong> = 208.00</td>
</tr>
</tbody>
</table>
Table 5.5 (continued)

<table>
<thead>
<tr>
<th>Query 4</th>
<th>Query 5</th>
<th>Query 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth =41-51</td>
<td>Depth =23-37</td>
<td>Depth =</td>
</tr>
<tr>
<td>Color =very pale brown</td>
<td>Color = dark red</td>
<td>Color =dark red</td>
</tr>
<tr>
<td>Sand =sandy</td>
<td>Sand =sandy clay</td>
<td>Sand =sandy clay</td>
</tr>
<tr>
<td>Strength =medium</td>
<td>Strength =moderate</td>
<td>Strength =sandy clay</td>
</tr>
<tr>
<td>subangular blocky</td>
<td>medium subangular</td>
<td>medium subangular</td>
</tr>
<tr>
<td>Moist =</td>
<td>blocky</td>
<td>blocky</td>
</tr>
<tr>
<td>Pores =</td>
<td>Moist =slightly sticky</td>
<td>Moist =slightly sticky</td>
</tr>
<tr>
<td>Clay =3.40</td>
<td>Pores =few fine pores</td>
<td>Pores =few fine pores</td>
</tr>
<tr>
<td>Silt =4.00</td>
<td>Clay =40.00</td>
<td>Clay =40.00</td>
</tr>
<tr>
<td>Sand=92.60</td>
<td>Silt =24.00</td>
<td>Silt =24.00</td>
</tr>
<tr>
<td>PH=8.00</td>
<td>Sand=36.00</td>
<td>Sand=36.00</td>
</tr>
<tr>
<td>EC=0.06</td>
<td>PH=7.17</td>
<td>PH=7.17</td>
</tr>
<tr>
<td>CA=3.57</td>
<td>EC=0.16</td>
<td>EC=0.16</td>
</tr>
<tr>
<td>Mg=0.51</td>
<td>CA=10.00</td>
<td>CA=10.00</td>
</tr>
<tr>
<td>Na=0.25</td>
<td>Mg=4.50</td>
<td>Mg=4.50</td>
</tr>
<tr>
<td>K=0.10</td>
<td>Na=1.28</td>
<td>Na=1.28</td>
</tr>
<tr>
<td>P2O5=4.00</td>
<td>K=0.93</td>
<td>K=0.93</td>
</tr>
<tr>
<td>K2O=49.00</td>
<td>P2O5=25.65</td>
<td>P2O5=25.65</td>
</tr>
<tr>
<td></td>
<td>K2O=195.00</td>
<td>K2O=195.00</td>
</tr>
</tbody>
</table>

Table 5.6 Number of Plants retrieved

<table>
<thead>
<tr>
<th>Query 1</th>
<th>Query 2</th>
<th>Query 3</th>
<th>Query 4</th>
<th>Query 5</th>
<th>Query 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Plants Retrieved</td>
<td>11</td>
<td>15</td>
<td>10</td>
<td>13</td>
<td>12</td>
</tr>
</tbody>
</table>