



GROUNDWATER INFORMATION SYSTEM

PHASE 1

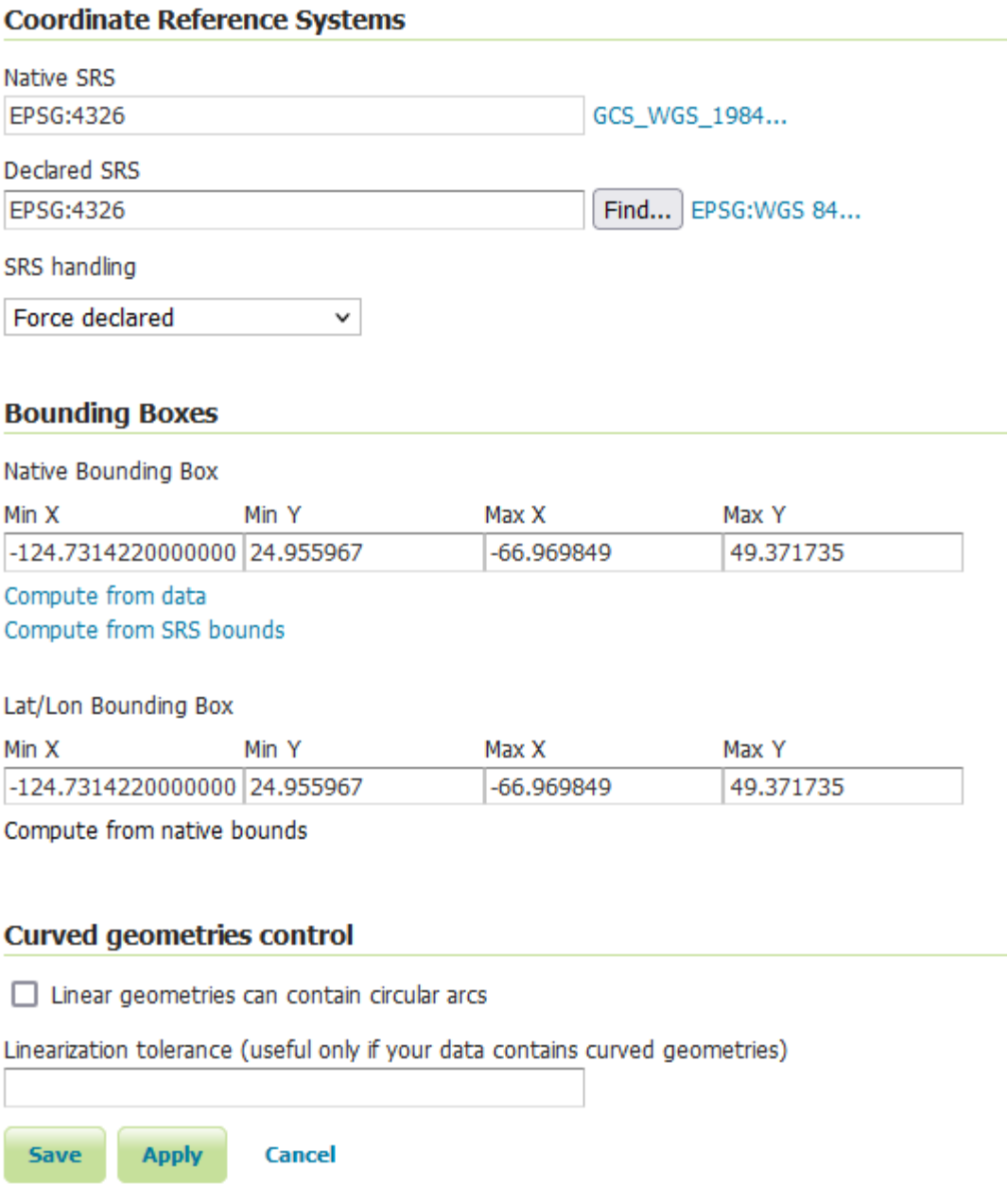
SYSTEM ADMINISTRATION

REFERENCE MANUAL

Table of Contents

PART 1 GENERAL OVERVIEW 7
OBJECTIVE AND SCOPE 8
DATA SHARING PROTOCOLS AND FORMATS 8
SOFTWARE 9
SYSTEM ARCHITECTURE & COMPONENTS 11
System components 11
Logical structure and Directory Hierarchy 12
PART 2 WEB APPLICATION 15
MODULES 16
EDITING MODULE -root directory 16
GRAPH MODULE – graphs directory 22
Aquifer_hydro.php 23
Monitoring Well Times Series (mw_timeseries.php) 25
Monitoring well – water level change (mw_decline.php) 25
Scatterplot - scatterplot_main.php 26
Compare WQ samples – radar2.php 26
Knowldege Base – knowledge_base.php, kb_downloads.php, kb-links.php, kb_admin.php 27
Table Views – views.php, view1.php, view2.php, view3.php, view4.php, view5.php, view6.php, view7.php 27
PART 3 - DATABASE DOCUMENTATION 34
INTRODUCTION 35
Postgres directories (Linux Ubuntu) 35
DATABASE DESIGN AND STRUCTURE 37
General Design 37
Table summary 39
Input forms 40
Spatial database 40
Time Series 41
Trigger Functions, Views, and Stored Procedures 42
Triggers 42
VIEWS 47
Indexes 54
Constraints 55
ACCESSING THE GWIS DATABASE 56
Introduction to the pgAdmin interface 59
DATABASE ADMINISTRATION 69
BACKUP & RESTORE 69
COPY COMMAND 70

EXCEL TEMPLATES TO CREATE CSV FILES ... 72
DELETING RECORDS ... 73
REGISTERED EDITORS AND THE USERS TABLE ... 74
CONFIGURATION FILES ... 75
TABLESPACE (optional advanced topic) ... 76
PART 4 – Geoserver ... 80
OVERVIEW ... 81
Prerequisites ... 82
OGC standards ... 82
Access to Administration ... 83
Workspaces ... 84
Data Directory ... 84
Stores ... 86



..... 89
LAYERS 90
Styles 91
SECURITY 93
95
SUMMARY OF WORKFLOW 95
Group Layers 96
PART 5 -ADDING DATA TO THE SYSTEM 97
Browser Forms for record-by-record data entry 98
Updating records using the web forms 104

ANNEX 1: 107
PROGRAMMING CODE 107
General Procedure 108
SQL 108
PHP 112
ADMIN MODULE 112
Graphs and Map Modules 141
JAVASCRIPT, AJAX and CDN Libraries 151

PART 1 GENERAL OVERVIEW

OBJECTIVE AND SCOPE

The main objective of this consultancy is to develop a comprehensive and functional HoA-GWIS to facilitate the sustainable management and use of shared groundwater resources within the IGAD region. Specifically, *the scope of the project regarding well information shall be restricted to transboundary aquifers shared by the member states*. If so desired at a later time, the system has the capacity to incorporate wells and other information at any location. In either case, emphasis is placed on analytical treatments able to detect trends, patterns, and groundwater resource assessment using dynamic data.

DATA SHARING PROTOCOLS AND FORMATS

The philosophy behind the system is universal information access regarding transboundary aquifers in the IGAD region. A special group of users, with the role of Editors, are required to first register with IGAD before acquiring the ability enter/modify data. General users can access the GWIS in a read-only mode – the ability to query the database for derived products only through the web application (direct connections to the database is reserved for IGAD/ICPAC), General users will not have the ability to insert, update, or delete records. In order to modify data in the database, special groups will be created, and further subdivided based on country. In other words, registered editors of a particular country can insert or update records of that country only.

The application that has been developed controls access based on the registration information, which is passed from the database into session variables.

Originally, an algorithm was written to validate the email domain of the registrant, however, many potential authorized users do not have institutional email accounts, so the algorithm has been commented out. Therefore, the creation of groups and permissions will be under the provenance of the database administrator, by inserting special codes into the *users* table. The records in the user table include username, password, agency, and activation (either 1-active, or 0-inactive) Instructions for this presented during training.

The member states (8 from IGAD plus 3 others from ICPAC) have been asked to consent to and sign a data sharing agreement, with the understanding that the data shared will only be wells within the transboundary aquifers and the raw records will not be viewed by others. Under this agreement, member states will contribute well and other information to populate the underlying database. Therefore, the focus of the system is on the assessment of the transboundary aquifers themselves, with analytical capabilities (through maps and

graphs) to facilitate resource management and collaborative decision making regarding these shared aquifers.

Once information has been submitted, IGAD will become the owner of such information, and users can only view derived products (not raw records).

The system created can handle inputs/outputs of the following formats in accordance with the Annex of the TOR:

1. Rasters of various format (Geotiff, Arcgrid, BIL, JPEG, PDF, Mr.SID, and others)
2. Vector data – shapefiles, geoJSON, geopackage
3. Tabular data (CSV, TXT, XLSX, Dat).

Well information, including location and completion dates, as well as hydrogeological parameters, water quality measurements, and construction details are held in the system, including monitoring wells and associated time series measurements. Rainfall and evapotranspiration for eight transboundary aquifers have also been processed into time series tables. The database can hold runoff data as well, but as of yet, this data has not been submitted.

SOFTWARE

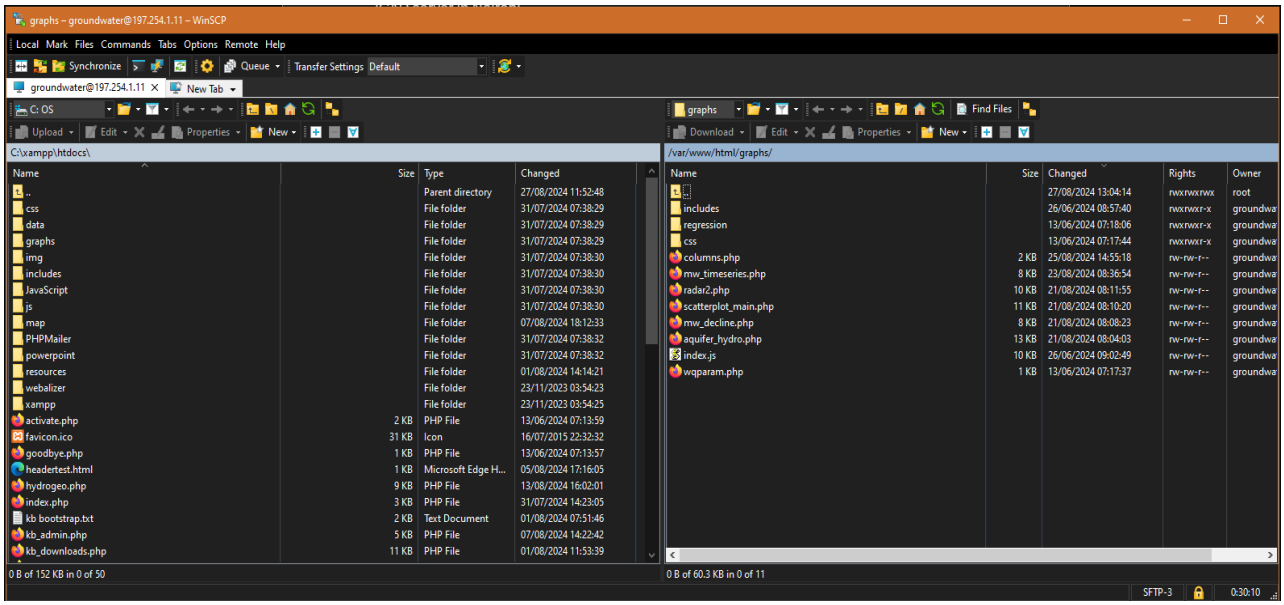
The philosophy of the GWIS is that all components that comprise the system are Free and Open Source (FOSS). These components are discussed in more detail in the next section.

The system has been made modular as much as possible. It has been assembled piece by piece to ensure a stepwise deployment and facilitate scalability for future development.

Development of the system has undergone three stages of testing. The first stage was conducted on a localhost domain on a single computer. Once the module performed adequately in this environment, it is tested again on a local area network (LAN) in home office. The final stage was the deployment of the module on the IGAD server in Nairobi.

External software to facilitate management of the system is also necessary for IGAD/ICPAC STAFF and system administrators. The main tool is a file transfer program to upload files to the server. This is required to replace or add new web pages for the application, although it is envisioned that this will only take place in phase 2 or in the event of migration to a new server (the web application for phase 1 is considered complete). However it is strongly encouraged to upload new map data for the Geoserver application, and this is discussed further in the Geoserver section.

The software for file transfer must be able to communicate through a Secure Shell (SSH) via SFTP. For Linux machines FileZilla is recommended, while Windows clients should use WinSCP. The example below demonstrates the use of WinSCP:



The panel on the left shows the file system of the local computer, while the panel on the right shows the file system of the remote server. To transfer files, simple drag and drop routines are used.

For database administrators, while there is a command-line utility (psql) for PostgreSQL, it is also recommended that a GUI client such as pgADmin OR Dbeaver be used.

SYSTEM ARCHITECTURE & COMPONENTS

In general terms the modules can be identified as follows:

- Login and editing system module
- Module designed to create charts and graphs presentable to the user.
- Module for Interactive web maps and GIS capabilities.
- Miscellaneous pages:

Knowledge base accessing downloadable resources and relevant links

Table views for assorted database queries that might be of interest

Web Map services page that relates to WMS, WFS, and WCS delivery

System components

The operating system by consensus has been deemed to be Linux Ubuntu server.

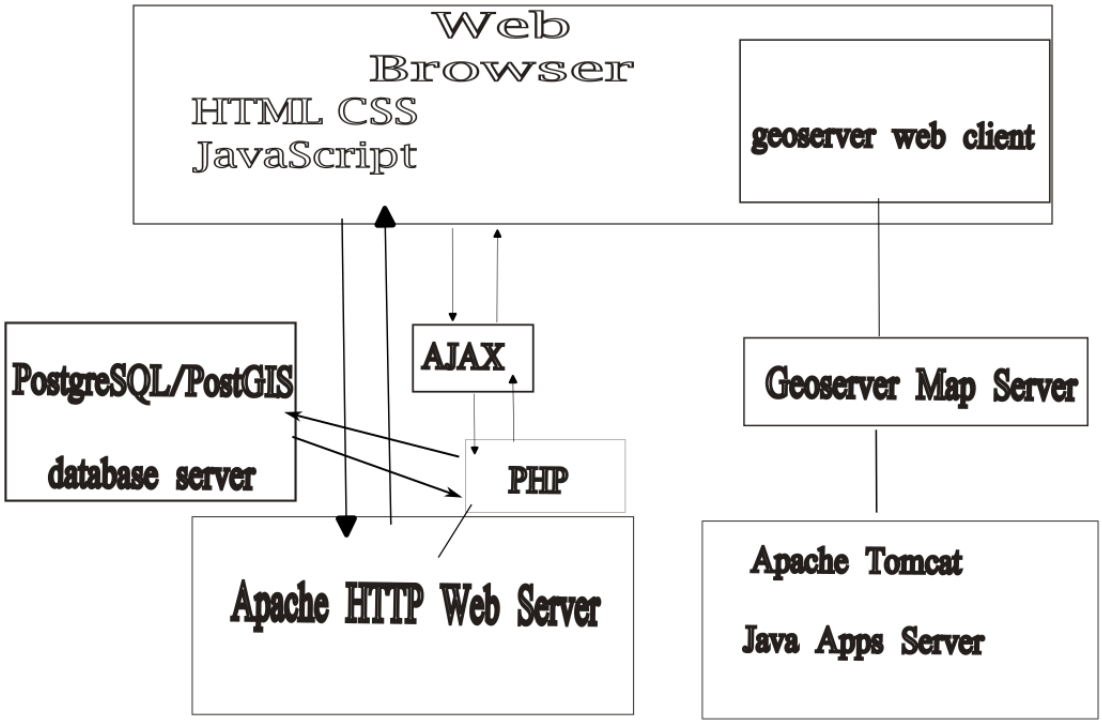
The web server, responsible for handling HTTP requests and delivering web pages to the client, is the Apache HTTP server 2.4. A Java server, Apache Tomcat, as well as a Java runtime environment is required to run the map server. *GeoServer* was chosen as the map server. This map server is described in detail in a separate section.

The Relational Database Management System (RDBMS) is PostgreSQL/Post GIS, by far the most powerful FOSS database system in use today. The PostGIS extension, with over 1,100 extra functions, allows for comprehensive spatial analysis needed for GIS functionality. This software can provide all the informational needs of the system.

Scripting Languages: On the server side there must be programming code to communicate with and request information from the database. The consultant has chosen PHP, as it has a proven track record – over 75% of current websites employ this code for their applications. For the client side, besides the required HTML and CSS files needed for presentation, JavaScript and its specialized libraries (e.g. Leaflet, Chart JS, Turf.js, and jQuery) are used for browser programming and allow user interaction.

Finally, a technology was implemented to ferry requests from the client to PHP, which in turn requests the database. This technology is termed AJAX, which performs this task without requiring the web page to do a full reload.

A schematic of the system is presented in Figure 1 below:



Logical structure and Directory Hierarchy

The root directory for Apache web server on an Ubuntu operating system is var/www/html. All web content must be contained within this directory.

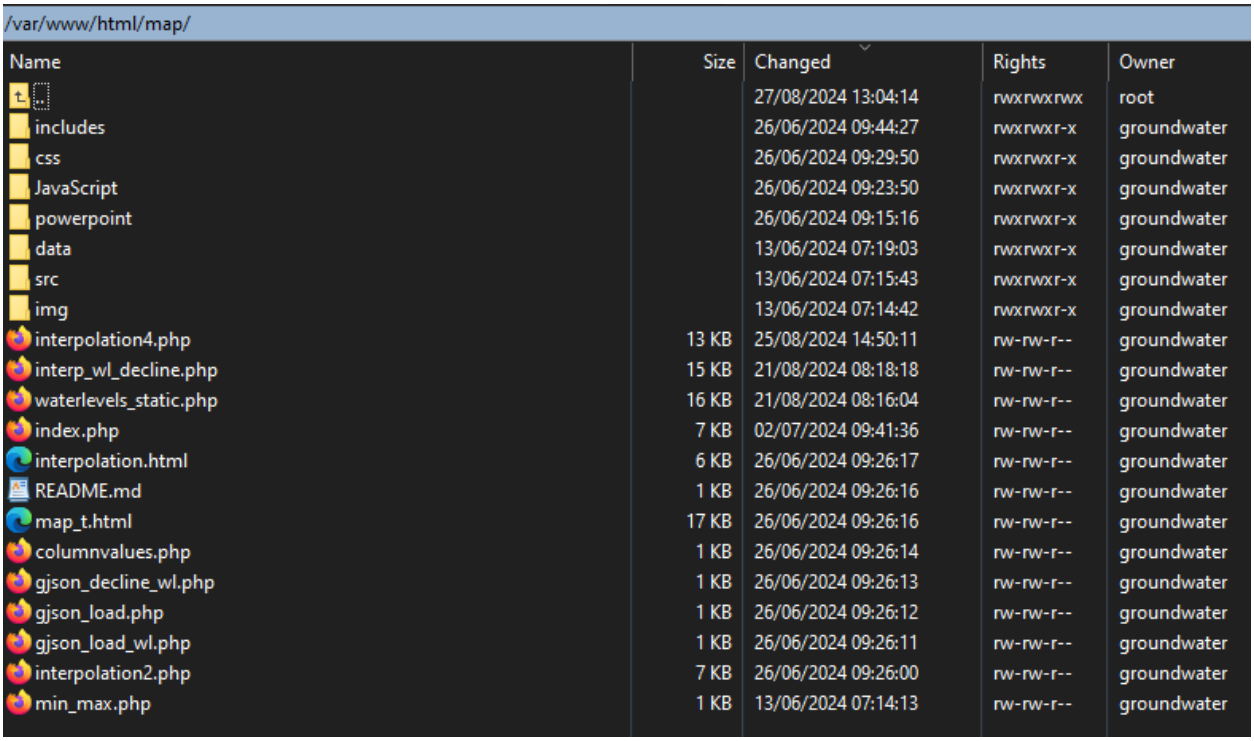
The system consists of three logical sections: administrative (login/registration and data input forms) in the root directory, and map, and graphs modules in their own subdirectories. A snapshot of the root directory is shown below:

/var/www/html/

Name	Size	Changed	Rights	Owner
..		06/06/2024 14:43:53	rwxr-xr-x	root
map		27/08/2024 13:04:14	rwxrwxr-x	groundwater
resources		07/08/2024 13:43:47	rwxrwxr-x	groundwater
graphs		27/06/2024 13:19:42	rwxrwxr-x	groundwater
includes		26/06/2024 08:17:55	rwxrwxr-x	groundwater
PHPMailer		21/06/2024 10:12:16	rwxrwxr-x	groundwater
js		14/06/2024 06:04:09	rwxrwxr-x	groundwater
JavaScript		13/06/2024 07:19:16	rwxrwxr-x	groundwater
css		13/06/2024 07:19:11	rwxrwxr-x	groundwater
update_rev2.php	4 KB	25/08/2024 13:49:32	rw-rw-r--	groundwater
update2.php	4 KB	25/08/2024 13:48:30	rw-rw-r--	groundwater
update_rev1.php	3 KB	25/08/2024 11:23:52	rw-rw-r--	groundwater
wellparent_add.php	8 KB	25/08/2024 10:13:45	rw-rw-r--	groundwater
WQ.php	8 KB	15/08/2024 07:05:38	rw-rw-r--	groundwater
hydrogeo.php	9 KB	13/08/2024 16:02:01	rw-rw-r--	groundwater
well_construct.php	12 KB	13/08/2024 15:43:46	rw-rw-r--	groundwater
review.php	7 KB	13/08/2024 14:16:32	rw-rw-r--	groundwater
views.php	10 KB	08/08/2024 13:55:24	rw-rw-r--	groundwater
view7.php	2 KB	08/08/2024 13:52:01	rw-rw-r--	groundwater
view6.php	2 KB	08/08/2024 13:32:21	rw-rw-r--	groundwater
kb_admin.php	4 KB	07/08/2024 14:23:53	rw-rw-r--	groundwater
update1.php	3 KB	06/08/2024 10:49:47	rw-rw-r--	groundwater
view5.php	2 KB	03/08/2024 15:56:10	rw-rw-r--	groundwater
view4.php	2 KB	03/08/2024 15:42:37	rw-rw-r--	groundwater
view3.php	2 KB	03/08/2024 14:28:06	rw-rw-r--	groundwater
view2.php	2 KB	03/08/2024 14:08:28	rw-rw-r--	groundwater
view1.php	2 KB	03/08/2024 14:04:53	rw-rw-r--	groundwater
wms.php	3 KB	01/08/2024 15:46:20	rw-rw-r--	groundwater
kb_links.php	5 KB	01/08/2024 13:48:14	rw-rw-r--	groundwater
knowledge_base.php	3 KB	01/08/2024 12:02:48	rw-rw-r--	groundwater
kb_downloads.php	11 KB	01/08/2024 11:53:39	rw-rw-r--	groundwater
login.php	5 KB	31/07/2024 14:44:10	rw-rw-r--	groundwater
logout.php	1 KB	31/07/2024 14:39:33	rw-rw-r--	groundwater
index.php	3 KB	31/07/2024 14:23:05	rw-rw-r--	groundwater
sendmail.php	5 KB	03/07/2024 10:39:15	rw-rw-r--	groundwater
register.php	6 KB	02/07/2024 15:25:19	rw-rw-r--	groundwater
phpinfo.php	1 KB	13/06/2024 07:46:57	rw-rw-r--	groundwater
activate.php	2 KB	13/06/2024 07:13:59	rw-rw-r--	groundwater
reset_1.php	3 KB	13/06/2024 07:13:55	rw-rw-r--	groundwater
reset_2.php	4 KB	13/06/2024 07:13:53	rw-rw-r--	groundwater

Most of these files belong to the Editing/Admin Module described later, but the Knowledge Base, Web Mapping Services, and Table Views pages are also located here. Also note the supporting folders: *resources, includes, js, JavaScript*, and *css* folders which many of the pages rely on. The map and graphs folders contain their respective modules which are described in the Web Application section.

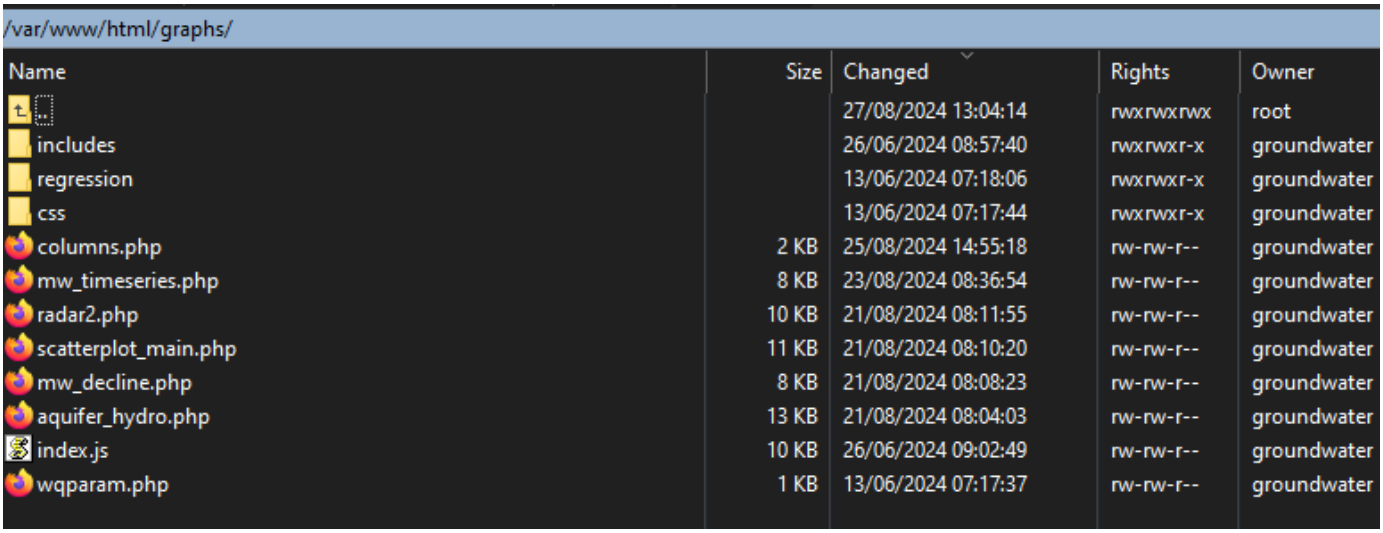
The map subdirectory is shown below:



/var/www/html/map/

Name	Size	Changed	Rights	Owner
..		27/08/2024 13:04:14	rwxrwxrwx	root
includes		26/06/2024 09:44:27	rwxrwxr-x	groundwater
css		26/06/2024 09:29:50	rwxrwxr-x	groundwater
JavaScript		26/06/2024 09:23:50	rwxrwxr-x	groundwater
powerpoint		26/06/2024 09:15:16	rwxrwxr-x	groundwater
data		13/06/2024 07:19:03	rwxrwxr-x	groundwater
src		13/06/2024 07:15:43	rwxrwxr-x	groundwater
img		13/06/2024 07:14:42	rwxrwxr-x	groundwater
interpolation4.php	13 KB	25/08/2024 14:50:11	rw-rw-r--	groundwater
interp_wl_decline.php	15 KB	21/08/2024 08:18:18	rw-rw-r--	groundwater
waterlevels_static.php	16 KB	21/08/2024 08:16:04	rw-rw-r--	groundwater
index.php	7 KB	02/07/2024 09:41:36	rw-rw-r--	groundwater
interpolation.html	6 KB	26/06/2024 09:26:17	rw-rw-r--	groundwater
README.md	1 KB	26/06/2024 09:26:16	rw-rw-r--	groundwater
map_t.html	17 KB	26/06/2024 09:26:16	rw-rw-r--	groundwater
columnvalues.php	1 KB	26/06/2024 09:26:14	rw-rw-r--	groundwater
gjson_decline_wl.php	1 KB	26/06/2024 09:26:13	rw-rw-r--	groundwater
gjson_load.php	1 KB	26/06/2024 09:26:12	rw-rw-r--	groundwater
gjson_load_wl.php	1 KB	26/06/2024 09:26:11	rw-rw-r--	groundwater
interpolation2.php	7 KB	26/06/2024 09:26:00	rw-rw-r--	groundwater
min_max.php	1 KB	13/06/2024 07:14:13	rw-rw-r--	groundwater

Finally, there is the graphs subdirectory:



/var/www/html/graphs/

Name	Size	Changed	Rights	Owner
..		27/08/2024 13:04:14	rwxrwxrwx	root
includes		26/06/2024 08:57:40	rwxrwxr-x	groundwater
regression		13/06/2024 07:18:06	rwxrwxr-x	groundwater
css		13/06/2024 07:17:44	rwxrwxr-x	groundwater
columns.php	2 KB	25/08/2024 14:55:18	rw-rw-r--	groundwater
mw_timeseries.php	8 KB	23/08/2024 08:36:54	rw-rw-r--	groundwater
radar2.php	10 KB	21/08/2024 08:11:55	rw-rw-r--	groundwater
scatterplot_main.php	11 KB	21/08/2024 08:10:20	rw-rw-r--	groundwater
mw_decline.php	8 KB	21/08/2024 08:08:23	rw-rw-r--	groundwater
aquifer_hydro.php	13 KB	21/08/2024 08:04:03	rw-rw-r--	groundwater
index.js	10 KB	26/06/2024 09:02:49	rw-rw-r--	groundwater
wqparam.php	1 KB	13/06/2024 07:17:37	rw-rw-r--	groundwater

Administration and maintenance – It is only important to note where to place files should it be desired to update existing files and/or to insert new ones. As for Apache web server, no administration is needed at this time, but in the future when traffic becomes heavy, there may be a need to fine tune the application for performance/security reasons. Administrators should consult the online documentation in that case. In summary, no administration or maintenance is required for the web application in normal circumstances.

PART 2 WEB APPLICATION

MODULES

The web application is divided into sections that parallel the directory structure as outlined in SECTION 1. These sections can be termed modules since they are independent and autonomous and therefore can be developed further without affecting the other parts.

Each module is self-contained in its own directory, with whatever resources and *include* files that are required, such as init.php to connect to the database and navigation bars.

The web pages also contain links to scripts, mainly JavaScript libraries and style sheets, which are obtained through content delivery networks (cdn):

```
<meta charset="utf-8">
    <link rel="stylesheet" href="https://code.jquery.com/ui/1.13.3/themes/base/jquery-ui.css">
    <script src="https://code.jquery.com/jquery-3.7.1.js"></script>
    <script type="text/javascript" src="https://cdn.jsdelivr.net/npm/chart.js/dist/chart.umd.min.js"></script>
    <script src="https://code.jquery.com/ui/1.13.3/jquery-ui.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/hammerjs@2.0.8"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/chartjs-plugin-zoom/2.0.1/chartjs-plugin-zoom.min.js"
```

Some scripts are also called locally, usually from a JavaScript directory on the server.

Each of the modules are briefly summarized in the pages that follow. The descriptions are not meant to be a line-by-line description of the code. If a review of the code is desired, it is suggested to download the web pages from the server and read them in a text editor (preferably a code editor).

EDITING MODULE -root directory

These pages are basically forms that send user inputs via POST methods to php scripts. The php code is tasked with sending insert queries to the database, thus adding new records. A user with editing privileges must first login where credentials are checked against a *user* table. Files in this module are linked to files in the module *includes* directory:

```
<?php include "includes/init.php" ?>
<?php include "includes/header.php" ?>
<?php include "includes/nav4.php" ?>
```

The **init .php** file creates the database connection and starts the session, while the **nav.php** file provides a navigation menu particular for this module (menu items related to record input). It also has an include statement linking to **the "php_functions.php"** file. The login page in particular makes heavy use of the functions in this file. Developers should refer to this document (php_function.php) to understand the various function calls.

Login page

Several variables are held in session variables:

```
$_SESSION['username'] = $username;
$country = get_country($username, $pdo);
$_SESSION['country'] = $country;
```

Note the call to the get_country() function (located in the php_functions.php file) which retrieves the country of the user in the users table. This is the country that will be placed into the well_parent table, thus ensuring that the user can only enter records for the country that he/she represents.

wellparent_add.php

Of particular note on this page is the calculation of well number:

```
$well_no = $date_completed . "-" . substr($country, 0, 2) . "-" . mt_rand(1, 9) . mt_rand(10, 50);
```

Which takes the date completed concatenated with the first 2 letters of the country name, and adds 2 sets of random numbers to create the well number such as '2006-01-22-Ke-377.' This well number is held in a session variable until the user chooses 'Add Well' from the menu to enter information for another well, when another well_no variable is generated.

Also, a function had to be created (because of a quirk in php) to handle missing values in the form -

```
convert_blank_strings_to_null($_POST, $recursive = false);
```

For all of the pages that insert records into the database, the SQL queries are wrapped inside try/catch blocks, and SQL injection is prevented using prepared statements. Here's an example of such code from wellparent_add.php:

```
try {
```

```
    $sql = "INSERT INTO well_parent (well_no, easting, northing, purpose, country,  admin1,  admin2,
       admin3, settlement, orig_wellno,  date_completed, coord_sys, elev, status)
       VALUES (:well_no, :easting, :northing, :purpose, :country,  :admin1,  :admin2, :admin3,
       :settlement, :orig_wellno, :date_completed, :coord_sys, :elev, :status)";
    $stmnt = $pdo->prepare($sql);
    $user_data = [
       ':well_no' => $well_no, ':easting' => $easting, ':northing' => $northing, ':purpose' => $purpose,  ':country' =>
$country,
       ':admin1' => $admin1, ':admin2' => $admin2, ':admin3' => $admin3, ':settlement' => $settlement,
':orig_wellno' => $orig_wellno,
       ':date_completed' => $date_completed, ':coord_sys' => $coord_sys, ':elev' => $elev, ':status' => $status
    ];
    $stmnt->execute($user_data);


    redirect('well_construct.php');
  } catch (PDOException $e) {


    echo "Error: " . $e->getMessage();
  }
```

If the insertion is successful, the user is directed to the next page with a form for the next table. In the above case, well_construct.php. As is stressed in the *Users Manual*, **the user must proceed through all four pages (by clicking the Add Record button)** in order to place a well_no into all the tables, regardless of whether any information is entered. This holds a place for the well in those tables, where information can be added in the future when it is obtained.

SQL statements wrapped in try/catch blocks are also used to provide values for <select> drop-down list – many examples of this are seen throughout the web application. In this example, if users enter coordinates into the form, they must specify a coordinate system by selecting from a list derived from the coordinate system table (crs_lookup) of the database. The default choice is WGS 84, the default for most GPS devices.

```
<select name="coord_sys" id="coord_sys" class='form-control'>
        <?php
```

```
if (!is_null($easting) || !is_null($northing))
    echo "<p> For coordinates you must enter a coordinate system/p>";
try {
    $result = $pdo->query("SELECT * FROM crs_lookup where countries like '%{$_SESSION['country']}%'
    OR countries like 'Entire%' ORDER BY system desc");
    foreach ($result as $row) {
        echo "<option value={$row['srid']}>{$row['system']}</option>";
    }
} catch (PDOException $e) {
    echo "Error: " . $e->getMessage();
}
?>
</select>
```

This information is required for a trigger function that converts text coordinates into geometry objects (described in the database section), and the conversion applied depends on the coordinate system.

well_construct.php

This page, and the other two that follow, have forms whose inputs are restricted through data validation, either JavaScript functions or HTML constraints.

As we've seen previously, SELECT input elements with drop-down lists restrict choices to values in a database table through the help of php code, as this one to select drilling method:

```
<select name='drilling_meth' id='drilling_meth' value='unknown'>
    <?php
    try {
        $result = $pdo->query("SELECT method FROM drilling_meth ORDER BY method desc");
        foreach ($result as $row) {
            echo "<option value={$row['method']}>{$row['method']}</option>";
        }
    } catch (PDOException $e) {
        echo "Error: " . $e->getMessage();
```

```
} ?>
 </select>
```

Such strategy ensures consistency, as users might input different text for the same method, such as cable-tool instead of percussion, DTH instead of hammer, or even input names of rigs, eg. Ingersoll Rand, DANDO 350, etc.

Onchange events fire javascipt functions to check that casing diameter is less than hole diameter, screen bottoms are lower than screen tops, but above drilling depth, etc. Invalid entries are not permitted, and the user is notified via an alert box.

```
function casing_val() {
    //Set the element variables
    let xElement = document.getElementById("casing_diam_mm");
    let yElement = document.getElementById("hole_diam_mm");
    let x = parseFloat(xElement.value); // parseFloat converts string to floating number
    let y = parseFloat(yElement.value);

    if (x > y) {
       xElement.value = ""; // use the element variable to set the value
       alert("casing diameter must be less than hole diameter. Try again.");
    }
 }
```

HTML constraints ensure *positive numbers* are entered into numeric fields.

<input **type="number" step="any" min=0** name="depth_dbg" id="depth_dbg" class="form-control" placeholder="Finished depth below ground" value="<?php echo $depth_dbg ?>">

hydrogeo.php.

Everything mentioned in the description of well_construct.php applies here as well.

WQ.php

On this form, data validation is performed by HTML constraints only. There is a difference in that maximum values are specified. Below, we see the maximum allowed value for chloride is 1500 mg/l.

<input type="number" step="any" min=0 **max=1500** name="cl_mgl" id="cl_mgl" tabindex="6" class="form-control" placeholder="Chloride mg/l" value="<?php echo $cl_mgl ?>">

These maximum values are thought to be in the highest range for freshwater, but they are subjective. Developers can change this value if users request this.

Review and Update

After information on the WQ form (the last form) has been entered and the Add Record button clicked the application presents the review page, where all the data entered for all tables can be reviewed. This is done through an SQL query that fetches the last record entered for each table.

Buttons on the bottom are wrapped inside anchor tags with various href values that can lead to the wellparent_add table to enter another well, to get out of editing and go to the Home page, or to an update page where the well number is kept in a session variable and placed in the where condition input box. This page, called *update_rev1*, allows the user to choose a table to modify a column from that table. The user is then redirected to *update_rev2* which offers 1) the columns to update, 2) an input text box for the new value, and 3) the where clause which is automatically filled with the current well-no. These pages rely on querying the PostgrSQL system view, *information_schema.Columns*.

Users can also update tables in a separate session (apart from adding new wells) by choosing the *Update well info* from the editing menu. These pages operate in the same way, except now the user is free to choose the column for the where clause.

The user manual makes clear how these pages are used.

GRAPH MODULE – graphs directory

All the pages in this module use the JavaScript library *Chart.js* to produce graphs in a browser document. Code is organized into 3 main blocks: data block, config block, and render block.

The data block contains the data to graph, and this format should be used to add additional datasets (discussed later)

```
{
        label: 'average et',
        data: avg_aet,
        backgroundColor: [
          'rgba(255, 26, 104, 0.2)',

        ],
        borderColor: [
          'rgba(255, 26, 104, 1)',

        ],
        borderWidth: 1,
        tension: 0.4
      }
```

For more details, consult the online documentation for Chart.js

Data is retrieved from the database using POST methods to server-side php scripts which send the results to the browser. This data is passed into javascript variables by use of the php json_encode function, as shown in the example below:

```
const rainfall = <?php echo json_encode($rainfall) ?>;
    const year = <?php echo json_encode($year) ?>;
    const et = <?php echo json_encode($et) ?>;
```

Aquifer_hydro.php

This page has data supplied by 2 tables, trans_ann_climate and trans_clime_monthly, which are described in the database section., and both are joined to the trans-aquifers table. The page shows both yearly totals and long-term monthly averages for rainfall and actual evapotranspiration for each aquifer in the database.

Data is retrieved from the queries through foreach loops:

```
foreach ($result as $row) {
        $avg_rainfall[] = $row['avg_rainfall'];
        $avg_aet[] = $row['avg_aet'];
        $month[] = $row['month'];
    }
```

The variables are then passed to JavaScript through the php json_encode function mentioned previously.

A common feature in many of the web pages is making database calls to populate a <select> dropdown list, as is shown here:

```
<select name="aquifer" id="aquifer2">
                    <option value="">--click here--</option>
                    <?php

                    try {
                        $result = $pdo->query("SELECT name, code FROM trans_aquifers;");
                        foreach ($result as $row) {
                            echo "<option value={$row['name']}>{$row['name']}</option>";
                        }
                    } catch (PDOException $e) {
                        echo "Error: " . $e->getMessage();
                    }
                    ?>
```

```
        </select>
```

Users can choose between line and bar charts, which is made possible by the changeChart function:

```
function changeChart(charttype) {

    myChart.config.type = charttype;
    myChart.update();
}
```

Which responds to the button onclick event.

Scalability

Obviously, there are components of the hydrological cycle which are absent and should be included, notably runoff, and perhaps soil moisture changes. The consultant did not do this as the data was not provided. To do so, one would first need to put this information into the database. This requires adding the appropriate columns to the two tables mentioned above and filling in values. This procedure is explained in the database section.

Then the foreach loop in the php code would have to be modified to include these columns.

e.g. $runoffl[] = $row['runoff'];

and a data block should be added into the *Chart.js* code for each variable (see above).

Aquifers needed to be added as the database expands. At the moment there are nine transboundary aquifers in the system overlying the borders of Kenya, Somalia, and Ethiopia (and parts of Sudan). For rainfall and Et, data needs to be processed from the CHIRPS datasets available on various sites of the internet, and MODIS satellite data respectively. Details of the acquisition and processing are dealt with in another part of this manual.

Future Enhancements – none foreseen.

Monitoring Well Times Series (mw_timeseries.php)

Again, we see a similar pattern that is shared by most of the graph pages – php code behind a form that retrieves the appropriate data from the database. For this page, data comes from the mw_data table, which is a time series of measurements taken in monitoring wells. Data comes back in the form of php associative arrays which are then converted into JavaScript arrays which can then be plotted through the Chart.js routines. The same *Chart.js* code structure appears in all the graph pages.

Scalablity

Updating the chart only depends on adding time series data to the mw-data table. New wells are created in the monitoring wells table when wells are added to the well_parent table with the p*urpose* field set to 'monitoring', accomplished via a trigger function (described in the database section).

Future Enhancements – none foreseen.

Monitoring well – water level change (mw_decline.php)

This page, besides also using data from the mw_data table, uses 2 special 'Views' called *mw-record* and *total_decline*. The view *total_decline* is a rather complicated sql query comprising of two Common Table Expressions that use specialized datetime manipulation and window functions to calculate the difference in the yearly average of water levels in a a well. mw_record calculates the start and end dates, and the period of record. This information is displayed in the browser along with the graph, the latter showing the cumulative difference between the level at a certain point and the value at the start of measurement.

The coding techniques used are similar to those already described.

Scalability

As these views are dynamically based on the monitoring_wells and mw-data tables, the same requirements for updating the ordinary time series applies:

Updating the chart only depends on adding time series data to the mw-data table. New wells are created in the monitoring wells table when wells are added to the well_parent table with the p*urpose* field set to 'monitoring', accomplished via a trigger function (described in the database section).

Future Enhancements – none foreseen.

Scatterplot - scatterplot_main.php

This graph differs from the others in that it makes two AJAX calls to a php script called columns.php, which queries a system view called information_schema.columns to retrieve all the columns in the database. This allows any variable to be plotted against any other variable.

Scalability

Since all of the data comes from the database, and is thus dynamic, scalability only entails adding data to the four main well tables.

Future Enhancements

There are two possible enhancements. One is to group the variables into sets, such as restricting the variables to a certain transboundary aquifer, or within a certain geographic region, rather than plotting the variables from ALL of the wells. This could be done by modifying the database query, but was not implemented because the lack of data makes testing impossible.

The second enhancement follows from the first. An interactive map can be used to choose the wells for plotting the variables. This requires complex coding, and the consultant was limited by time and resources to apply this approach, but it should definitely be considered for phase two.

It would also be convenient if labels associating the points with individual wells could be displayed on the graph. The charting library, Chart.js, cannot implement this. The consultant looked at an alternative, Apache e-Charts, but this limitation also arose.

Compare WQ samples – radar2.php

This is not much different from the other charts in terms of coding. Data is retrieved from the database using POST methods to server-side php scripts which send the results to the browser. This data is passed into JavaScript variables by use of the php json_encode function.

As with the scatterplot, an interactive map might be used to choose the wells for plotting the variables. Geochemical patterns often follow spatial trends and could potentially distinguish different aquifer bodies. Again, this could be considered for phase two.

Knowldege Base – knowledge_base.php, kb_downloads.php, kb-links.php, kb_admin.php

Since this section is outside of the other modules, the associated pages are located in the root directory.

The coordinating page is knowledge_base.php with introductory content in the main part of the page and a side panel dividing the knowledge base into 3 sections. The first section, Downloadable Resources, navigates to the kb_downloads.php, which contains links to downloadable reports. The second item in the panel is Useful Links, which is a page with relevant hypertext links to other websites. The third item directs to a page with resources for administrators and registered editors, many of which pertain to adding data to the GWIS.

All of the downloadable items are contained within the resources folder in the root directory.

Scalability – With a knowledge of html, new links can easily be added, and any resources can be uploaded to the resources directory using the sftp software mentioned in the first part of this report.

Future Enhancements – none foreseen.

Table Views – views.php, view1.php, view2.php, view3.php, view4.php, view5.php, view6.php, view7.php

The main webpage which is displayed is views.php, which contains buttons that fire associated AJAX calls to the other php files, each of which echo out a table with column headers and data obtained from specific Views in the database. At the moment, the numbered view pages correspond to 7 different views, depending on the button clicked. For example, the button "Aquifer abstraction" fires this AJAX call:

```
<script>
        $("#aqabsview-submit").click(function() {

          $.ajax({
            url: 'view1.php',
            type: 'POST',
            // data: { view: "aqabsview" },
            success: function(response) {
              $("#resulttable").html(response);
```

```
            },
            error: function(xhr, status, error) {
                $("#resulttable").html("ERROR: " + ERROR);
            }
        })
    })
    </script>
```

The url parameter is the name of the php file that is invoked, in this case 'view1.php,' which contains the following code to format the table and return the query results:

```
try {
    $sql = "SELECT * FROM aq_abstract";
    $result = $pdo->query($sql);
    echo "<h3>Total abastraction by Transboundary Aquifer</h3><br><hr><br>";

    $returntable = "<table class='table table-striped table-hover'>";
    $row_h = $result->fetch();



    $returntable .= "<tr class= 'tblHeader'>";
    foreach ($row_h as $key => $val) {
        $returntable .= "<th>{$key}</th>";
    }
    $returntable .= "</tr>";

    $returntable .= "<tr>";
    foreach ($row_h as $key => $val) {
        $returntable .= "<td>{$val}</td>";
    }
    $returntable .= "</tr>";
    foreach ($result as $row) {
```

```
        //$returntable .= "<tr>";

        foreach ($row as $key => $val) {
            $returntable .= "<td>{$val}</td>";
        }
        $returntable .= "</tr>";
    }
    foreach ($result as $row) {
        $returntable .= "<tr>";
        foreach ($row as $key => $val) {
            $returntable .= "<td>{$val}</td>";
        }
        $returntable .= "</tr>";
    }
    $returntable .= "</table>";
    echo $returntable;
} catch (PDOException $e) {
    echo $e->getMessage();
}
```

The views themselves are formed from SQL queries of the database and are given below:

View1 – aq_abstract

```
SELECT ta.name AS aquifer,
    count(wp.well_no) AS "total number of wells",
    sum(h.discharge_m3_per_hr) * 180::numeric AS "annual total abstaction"
   FROM well_parent wp
     JOIN hydrogeo h ON h.well_no::text = wp.well_no::text
     JOIN trans_aquifers ta ON st_covers(ta.geom, wp.geom)
  GROUP BY ta.name;
```

View 2 - v_product_hydrogeo_2

```
SELECT wp.well_no,
    wp.orig_wellno,
    h.discharge_m3_per_hr,
    h.specific_capacity,
    h.water_strike_1_dbg,
    kse.hydrogeo_u AS "BGS Hydrogeological unit"
   FROM well_parent wp
     JOIN hydrogeo h ON wp.well_no::text = h.well_no::text
     JOIN kesomet_hydrogeo kse ON st_covers(kse.geom, wp.geom)
  ORDER BY h.discharge_m3_per_hr DESC NULLS LAST;
```

View 3 - v_well_hydrogeo

```
SELECT wp.well_no,
    wp.orig_wellno,
    h.lithology1_descrip AS "lithology at screen 1",
    h.lithology2_descrip AS "lithology at screen 2",
    h.lithology3_descrip AS "lithology at screen 3",
    kse.hydrogeo_u AS "BGS Hydrogeological unit",
    ta.name AS aquifer
   FROM well_parent wp
     JOIN hydrogeo h ON wp.well_no::text = h.well_no::text
     JOIN kesomet_hydrogeo kse ON st_covers(kse.geom, wp.geom)
     JOIN trans_aquifers ta ON st_covers(ta.geom, wp.geom)
  ORDER BY h.lithology1_descrip DESC NULLS LAST;
```

View 4 – wq_error

```
WITH wqsums AS (
        SELECT wq.well_no,
           wq.sample_no,
           wq.ca_meql + wq.mg_meql + wq.na_meql + wq.k_meql AS sumcat,
           wq.cl_meql + wq.hco3_alk_meql + wq.so4_meql AS suman,
           wq.ca_meql + wq.mg_meql + wq.na_meql + wq.k_meql + wq.cl_meql + wq.hco3_alk_meql + wq.so4_meql
AS tdi
          FROM wq
       )
 SELECT well_no,
    sample_no,
    sumcat AS "total cations",
    suman AS "total anions",
    tdi,
    (sumcat - suman) / tdi * 100::numeric AS "percent error"
   FROM wqsums;
```

View 5 - v_conjunctive

```
SELECT wp.well_no,
    wp.orig_wellno,
    st_distance(wp.geom::geography, ksm.geom::geography) / 1000::numeric::double precision AS "nearest distance
to stream km",
    ta.name AS aquifer,
    ksm.catch_skm AS "catchment area",
    ta.annual_rainfall_mm - ta.annual_et AS "annual excess precipitation mm",
    ((ta.annual_rainfall_mm - ta.annual_et) * 1000)::numeric::double precision * ksm.catch_skm AS "annual excess
precipitation cubic meters"
   FROM well_parent wp
     JOIN trans_aquifers ta ON st_covers(ta.geom::geography, wp.geom::geography)
     JOIN kesomet_rivers ksm ON st_dwithin(ksm.geom::geography, wp.geom::geography, 5000::double precision)
```

```
 ORDER BY (st_distance(wp.geom::geography, ksm.geom::geography) / 1000::numeric::double precision);
```

Please note that this is a very complicated query and may take over a minute to run!

View 6 - v_elev_scr

```
SELECT wp.well_no,
    wp.orig_wellno,
    st_distance(wp.geom::geography, ksm.geom::geography) / 1000::numeric::double precision AS "nearest distance to stream km",
    ta.name AS aquifer,
    ksm.catch_skm AS "catchment area",
    ta.annual_rainfall_mm - ta.annual_et AS "annual excess precipitation mm",
    ((ta.annual_rainfall_mm - ta.annual_et) * 1000)::numeric::double precision * ksm.catch_skm AS "annual excess precipitation cubic meters"
   FROM well_parent wp
     JOIN trans_aquifers ta ON st_covers(ta.geom::geography, wp.geom::geography)
     JOIN kesomet_rivers ksm ON st_dwithin(ksm.geom::geography, wp.geom::geography, 5000::double precision)
  ORDER BY (st_distance(wp.geom::geography, ksm.geom::geography) / 1000::numeric::double precision);
```

View 7 - v_ws_elev

```
SELECT wp.easting,
    wp.northing,
    wp.elev - h.water_strike_1_dbg AS elevation_first_water_strike,
    wp.elev - h.water_strike_2_dbg AS elevation_second_water_strike,
    h.discharge_m3_per_hr
   FROM well_parent wp
     JOIN hydrogeo h ON h.well_no::text = wp.well_no::text;
```

Scalability

Concerning the data, the views are dynamic and are updated automatically as records are added.

Future Enhancements

New views may be added according to feedback from users. This entails formulating the SQL queries to produce the results desired. Then what is needed is to make a copy of one of the php scripts, replacing the name of the view in the Select statement with the new view. Copying and pasting the AJAX calls and the HTML for the buttons would have to be done in the main views page to accommodate the new views. If the views become too many, it may be inconvenient as regards scrolling, so a new page could be created, for example views2.php, to hold the new views.

PART 3 - DATABASE DOCUMENTATION

INTRODUCTION

The relational database management server (RDMS) the GWIS uses is PostgreSQL version 16. It includes the extensions plpgsql, postgis, tablefunc, and postgis_raster. The purposes of extensions are as follow:

plpgsql – this is the main programming language for writing database functions. All the tirgger functions in the gwis database are written in this language.

tablefunc – With this function, pivot table queiries can be created

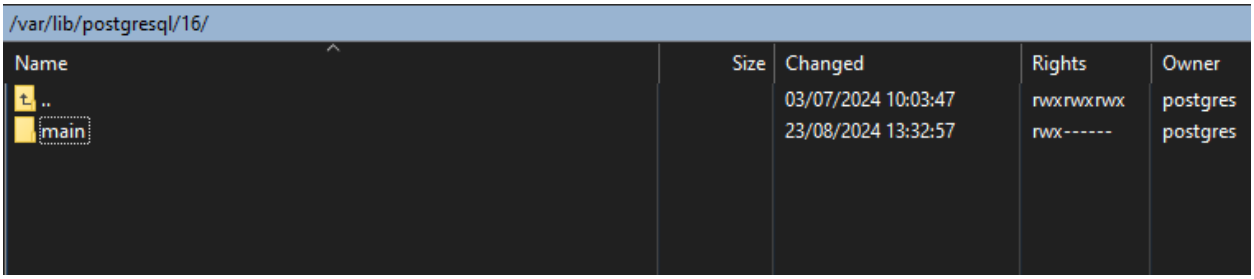
postgis – enables spatial queries

postgis-raster – allowing storing of raster data.

The database created for the IGAD groundwater information system is named **'gwis.'** This database has nearly 40 tables and over 30 views.

Postgres directories (Linux Ubuntu)

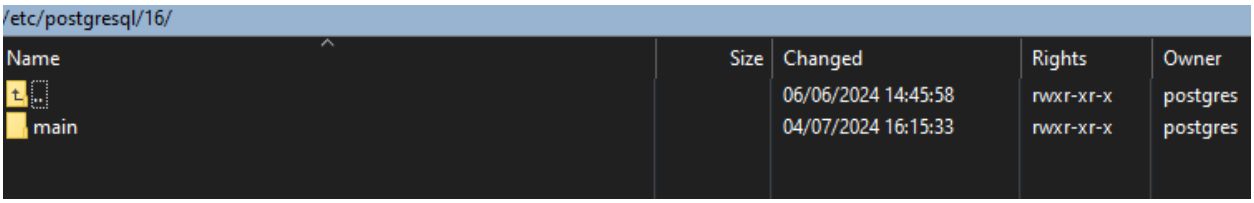
The postgres software is installed in the var/lib/postgresql directory, specifically in the directory called 16



The main directory is where the executable are, but only the postgres account can access it, unless sudo privileges are invoked: groundwater@gwis:/var/lib/postgresql/16$ **sudo ls -l main**

```
groundwater@gwis:/var/lib/postgresql/16$ sudo ls -l main
total 20
drwx------ 6 postgres postgres   46 Jun 12 19:41 base
drwx------ 2 postgres postgres 4096 Aug 23 06:33 global
drwx------ 2 postgres postgres    6 Jun  6 07:45 pg_commit_ts
drwx------ 2 postgres postgres    6 Jun  6 07:45 pg_dynshmem
drwx------ 4 postgres postgres   68 Aug 23 06:37 pg_logical
drwx------ 4 postgres postgres   36 Jun  6 07:45 pg_multixact
drwx------ 2 postgres postgres    6 Jun  6 07:45 pg_notify
drwx------ 2 postgres postgres    6 Jun  6 07:45 pg_replslot
drwx------ 2 postgres postgres    6 Jun  6 07:45 pg_serial
drwx------ 2 postgres postgres    6 Jun  6 07:45 pg_snapshots
drwx------ 2 postgres postgres    6 Aug 23 06:32 pg_stat
drwx------ 2 postgres postgres    6 Jun  6 07:45 pg_stat_tmp
drwx------ 2 postgres postgres   18 Jun  6 07:45 pg_subtrans
drwx------ 2 postgres postgres    6 Jun  6 07:45 pg_tblspc
drwx------ 2 postgres postgres    6 Jun  6 07:45 pg_twophase
-rw------- 1 postgres postgres    3 Jun  6 07:45 PG_VERSION
drwx------ 3 postgres postgres   92 Aug  3 09:51 pg_wal
drwx------ 2 postgres postgres   18 Jun  6 07:45 pg_xact
-rw------- 1 postgres postgres   88 Jun  6 07:45 postgresql.auto.conf
-rw------- 1 postgres postgres  130 Aug 23 06:32 postmaster.opts
-rw------- 1 postgres postgres  114 Aug 23 06:32 postmaster.pid
```

There is a parallel directory structure in etc/postgresql

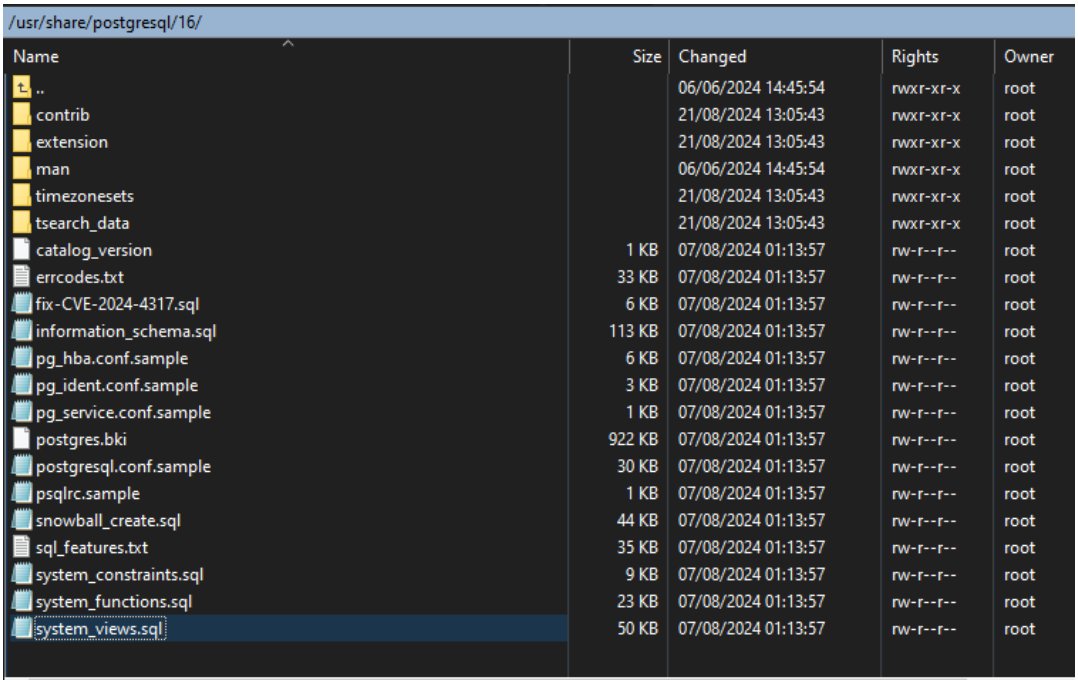


But the contents of the main file are slightly different:

```
groundwater@gwis:/etc/postgresql/16$ sudo ls -l main
[sudo] password for groundwater:
total 64
drwxr-xr-x 2 postgres postgres     6 Jun  6 07:45 conf.d
-rw-r--r-- 1 postgres postgres   315 Jun  6 07:45 environment
-rw-r--r-- 1 postgres postgres   143 Jun  6 07:45 pg_ctl.conf
-rw-r----- 1 postgres postgres  5924 Jun 18 00:54 pg_hba_backup.conf
-rw-r----- 1 postgres postgres  5922 Jul  4 09:15 pg_hba.conf
-rw-r----- 1 postgres postgres  2640 Jun  6 07:45 pg_ident.conf
-rw-r--r-- 1 postgres postgres 29939 Jul  4 09:13 postgresql.conf
-rw-r--r-- 1 postgres postgres   317 Jun  6 07:45 start.conf
```

Two important files are in here: **pg_hba.conf** and **postgresql.conf**

Also in/usr/share/postgresql/16/



These are mainly system files.

DATABASE DESIGN AND STRUCTURE

General Design

Despite the fact that PostgreSQL is an enterprise system which can work in multiuser envronments, the implementation of the database has only one user – the application – which takes on the role of a postgres superuser. End users do not and cannot make direct connections to the database. Therefore, it was decided that the execution of transactions was not a crucial requirement, as it is doubtful that more than one user will be writing to a particular table at the same time. Only a few designated users will be able to edit records.

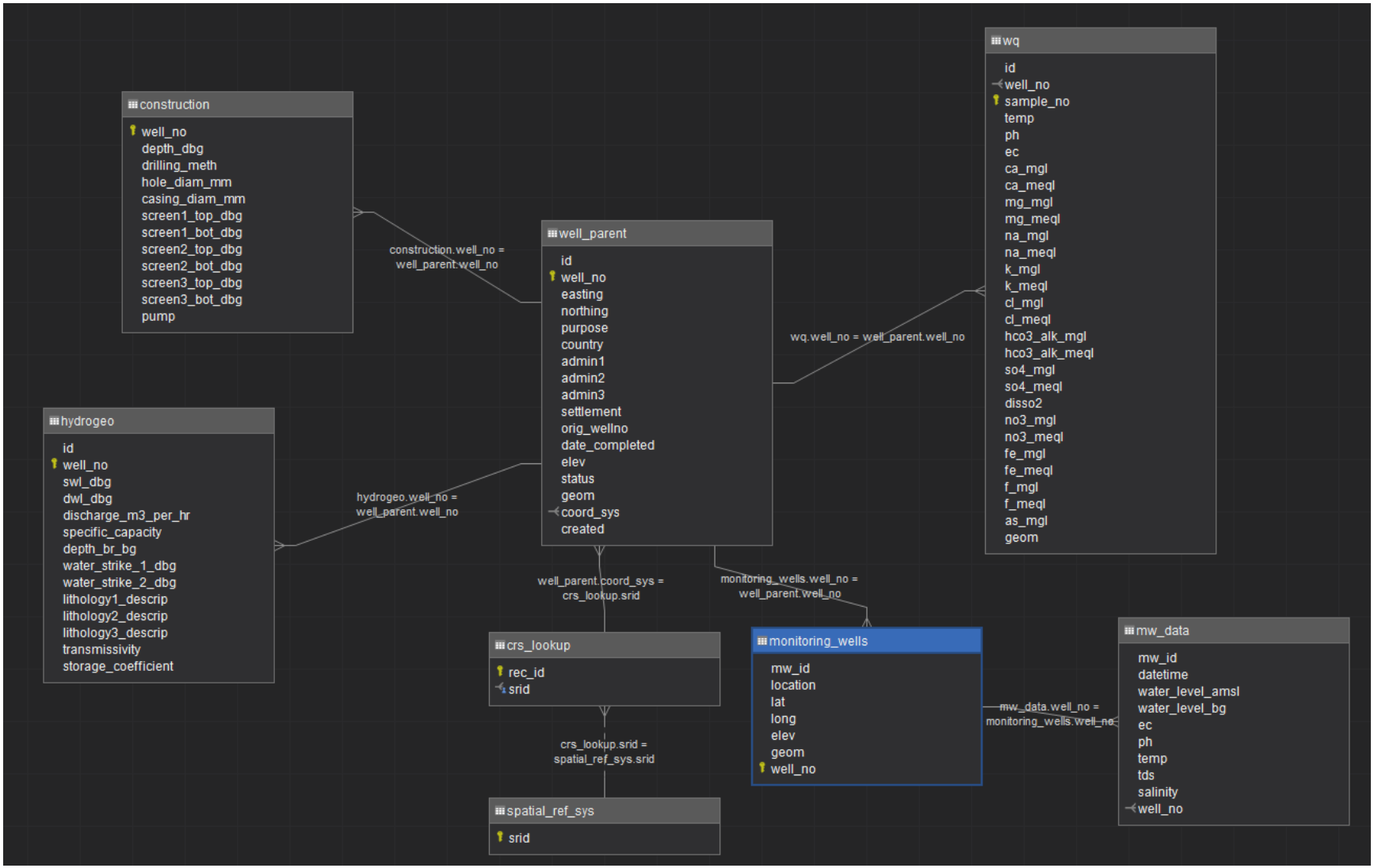
Unlike a typical business model, where a table is an object with attributes for that object, linked to other objects with their own attributes, and unlike the 2011 design, we have only one central object in the parent table that has dependents – the WELL. In this structure each record in the table is linked to a record representing the *same object* in another table denoted by the same primary key, which also acts as a foreign key. These other tables do not represent different objects but describe a different set of attributes

The core design incorporates a star-like topology. The center table can be considered as the parent to the other (child) tables. The fact that primary keys, which automatically have an index on them, also serve as foreign keys, make queries much faster without the overhead of creating additional indexes, particularly for joins using the primary keys.

There are many other tables which are outside the star topology not directly related to wells, including shapefiles. Some tables have a formal SQL relationship (foreign key constraints) such as a station table linked to a measurement table. The other chief relationship is spatial, in which entities are linked by their geometry columns through spatial queries this is discussed further in the section *Spatial database*.

Also included are geographical entities such as administrative/political boundaries, and the digitized extents of transboundary aquifers. Again, these entities can be linked to others through spatial queries.

The entity Relationship Diagram (ERD) of the central part of the database, the core design centered around wells, is shown on the next page:


construction
well_no
depth_dbg
drilling_meth
hole_diam_mm
casing_diam_mm
screen1_top_dbg
screen1_bot_dbg
screen2_top_dbg
screen2_bot_dbg
screen3_top_dbg
screen3_bot_dbg
pump
construction.well_no = well_parent.well_no
well_parent
id
well_no
easting
northing
purpose
country
admin1
admin2
admin3
settlement
orig_wellno
date_completed
elev
status
geom
coord_sys
created
wq
id
well_no
sample_no
temp
ph
ec
ca_mgl
ca_meql
mg_mgl
mg_meql
na_mgl
na_meql
k_mgl
k_meql
cl_mgl
cl_meql
hco3_alk_mgl
hco3_alk_meql
so4_mgl
so4_meql
disso2
no3_mgl
no3_meql
fe_mgl
fe_meql
f_mgl
f_meql
as_mgl
geom
wq.well_no = well_parent.well_no
hydrogeo
id
well_no
swl_dbg
dwl_dbg
discharge_m3_per_hr
specific_capacity
depth_br_bg
water_strike_1_dbg
water_strike_2_dbg
lithology1_descrip
lithology2_descrip
lithology3_descrip
transmissivity
storage_coefficient
hydrogeo.well_no = well_parent.well_no
well_parent.coord_sys = crs_lookup.srid
monitoring_wells.well_no = well_parent.well_no
crs_lookup
rec_id
srid
crs_lookup.srid = spatial_ref_sys.srid
spatial_ref_sys
srid
monitoring_wells
mw_id
location
lat
long
elev
geom
well_no
mw_data.well_no = monitoring_wells.well_no
mw_data
mw_id
datetime
water_level_amsl
water_level_bg
ec
ph
temp
tds
salinity
well_no

Note that the crs_lookup table exists only for the purpose of inputting coordinates through the web application and csv files. The well_parent table has a foreign key, **srid**, that links to this table, which in turn is a foreign key linked to the PostGIS system table spatial_ref_sys. These relationships are required to convert text strings of easting and northing to geometry objects through a trigger function (see section on triggers)

The other relationship which relies on foreign key constraints are measuring stations (monitoring wells, rain gauges, river discharge stations, etc.) and their associated time series data. Below is an example of the monitoring wells table and their table of measurements:

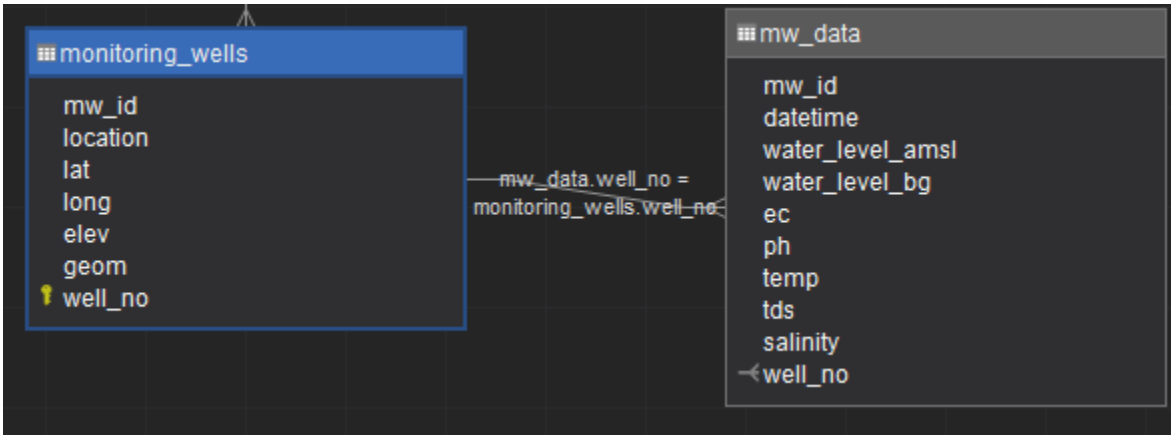


Table summary

A short description of each column in every table is given in the data dictionary section. In this section, some significant columns are mentioned.

well_parent – this is the center table, with fields representing location, elevation, date of completion, function and use, and the original borehole number from the member state's records. The key identifier of the system, the well_no, is generated by the PHP code behind the input form (see next section) and is shared with the other tables. As mentioned previously, they serve as the primary key as well as a foreign key in other tables (similar to the source id in the 2011 database).

construction – includes attributes related to construction of the well. Elevation is repeated here, and if input by the user to the primary table (well_parent) it is passed by a session variable and used to calculate elevation of well depth and screen levels, using a formula *elevation – depth below ground*. Note that the well number serves as a primary key and a foreign key linking back to the parent table, as is the case for all the child wells within the star topology.

hydrogeo - elevation should be the same value as that of the well_parent and construction tables and if input by the user to the primary table (well_parent) it is passed by a session variable and used to calculate elevations of water levels and strikes. Specific discharge is calculated from the

pumping rate divided by the difference between the dynamic water level and the static water level (drawdown). Lithologies are the geological formations corresponding to screen levels in the construction table (can be compared by using an SQL join with both tables).

wq – water quality (hydrochemistry) – In this table there are many generated columns which convert concentrations to milliequivalents per liter, necessary for analysis and the creation of graphs and diagrams. Computation of the total dissolved ions and the analytical error of the sample is done through a view (see later section).

Trans_aquifers – this is a table to hold the spatial and non-spatial attributes of the transboundary aquifers.

As mentioned previously, there are paired table relationships for measuring stations and their time series data with a code field that links them together (e.g. hydromet_station – temperature, evaporation, etc., and river gauge and rain gauge stations tied to river flows and rainfall respectively).

Input forms

Input forms have been developed. A description of some of the key programming of these forms was given in the first part of this manual, and a detailed description including screenshots is given in the *User Manual*. There is some automation involved in generating the IGAD well number. The proposed method is to concatenate the date, with a 2-letter abbreviation of the country name, and then a random number, each part separated by a dash (e.g. 2017-11-29-So-843). Also, well number and elevation are passed through session variables to other forms that need these values. Validation is done using both JavaScript (client-side), and PHP (server-side).

For bulk loading of many records, an Excel template was created. Each tab must be saved as a csv file and loaded via the COPY command, described in a later part of the section.

Spatial database

What is the relationship between wells and administrative units? Administrative units **contain** wells. And what would be a common question concerning a rain gauge and a well – how far away they are from each other, i.e. **distance**. This means that the relationship is purely spatial and that any tables outside those of the wells star topology do not require a foreign key link to any of the wells tables. Note the 'geom' column at the end of each table's list of columns. This is how the tables will interact using spatial queries (through their geometry columns). These columns are necessary to put them on a map. The foreign keys are only used for non-spatial queries.

One challenge is that many IGAD member countries use different coordinate reference systems (crs). In the well_parent table, there are fields for the user to input easting (x) and northing (y) coordinates, as well as a field for inputting the coordinate system they are in (a crs lookup table is provided for this purpose). A trigger uses these text inputs to create a binary geometry object, and provides for conversion to WGS84, which is the coordinate system the application uses (see section on Trigger Functions, Stored Procedures and Views).

Time Series

PostgreSQL can handle time series data using WINDOW functions, particularly LAG, LEAD, moving averages, and pivot tables (CROSSTAB). Some views have already been produced that extract parts of the date, calculate intervals, aggregate the data (including statistical aggregates), and adjust the granularity of the time series to fit the information sought (see next section).

Rainfall data for Kenya and Ethiopia were given to the consultant, but there were problems with these datasets that will be explained later. The consultant downloaded 40 years of rain over the transboundary aquifers from the CHIRPS database, and evapotranspiration data (since year 2000) was from acquired by the MODIS sensor aboard the Terra satellite.

Trigger Functions, Views, and Stored Procedures

Triggers

As mentioned previously, the easting (x) and northing (y)coordinates input by the user are converted from text strings into binary geometries using a trigger function, which is called by a trigger that fires upon insert or update. The code is shown below:

```
CREATE OR REPLACE FUNCTION public.fn_make_pointgeom()
  RETURNS trigger
  LANGUAGE 'plpgsql'
  COST 100
  VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN

        IF      NEW.coord_sys = 4326 then
                NEW.geom = ST_Point(NEW.easting::float, NEW.northing::float, 4326);
                ELSE
                NEW.geom = ST_Transform(ST_Point(NEW.easting::float, NEW.northing::float,
NEW.coord_sys), 4326);
                END IF;
        RETURN NEW;
```

```
END;

$BODY$;


CREATE OR REPLACE TRIGGER tr_make_pointgeom

BEFORE INSERT OR UPDATE

  ON public.well_parent

  FOR EACH ROW

  EXECUTE FUNCTION public.fn_make_pointgeom();
```

A second trigger function fires when the purpose field is filled with the value of ‘monitoring’,

```
CREATE OR REPLACE FUNCTION public.fn_monitoring_well_insert()

  RETURNS trigger

  LANGUAGE 'plpgsql'

  COST 100

  VOLATILE NOT LEAKPROOF

AS $BODY$


begin

--perform pg_sleep(1);

        insert into monitoring_wells (mw_id, location, lat, long, elev, geom, well_no )
```

```
select well_parent.orig_wellno, well_parent.settlement,

well_parent.northing, well_parent.easting, well_parent.elev, well_parent.geom,

well_parent.well_no from well_parent

where purpose='monitoring'

and created = current_timestamp;



--extract(minutes from created)= extract(minutes from localtimestamp);



return new ;



end;

$BODY$;



CREATE OR REPLACE TRIGGER tr_monitoring_well_insert

  AFTER INSERT OR UPDATE

  ON public.well_parent

  FOR EACH ROW

  EXECUTE FUNCTION public.fn_monitoring_well_insert();
```

Another trigger function is for auditing purposes, to track changes made to tables. At the moment, the trigger is attached to only the well_parent table, but it can and will be attached to most of the tables in the database. Whenever an insert, update or delete is performed, the old record is stored in a column of JSON format. The operation, user and time of the changes are also recorded. The

following is the CREATE script for the audit log table, the function executed, and the trigger that calls the function:

```
CREATE TABLE IF NOT EXISTS public.audit_log

(

  username text COLLATE pg_catalog."default",

  event_time_utc timestamp without time zone,

  table_name text COLLATE pg_catalog."default",

  operation text COLLATE pg_catalog."default",

  before_value json,

  after_value json

)



CREATE OR REPLACE FUNCTION public.audit_trigger()

  RETURNS trigger

  LANGUAGE 'plpgsql'

  COST 100

  VOLATILE NOT LEAKPROOF

AS $BODY$

DECLARE

old_row json := NULL;
```

```
new_row json := NULL;
BEGIN
IF TG_OP IN ('UPDATE','DELETE') THEN
old_row = row_to_json(OLD);
END IF;
IF TG_OP IN ('INSERT','UPDATE') THEN
new_row = row_to_json(NEW);
END IF;
INSERT INTO audit_log(
username,
event_time_utc,
table_name,
operation,
before_value,
after_value
) VALUES (
session_user,
current_timestamp AT TIME ZONE 'UTC',
TG_TABLE_SCHEMA || '.' || TG_TABLE_NAME,
TG_OP,
```

```
old_row,
new_row
);
RETURN NEW;
END;
$BODY$;


CREATE OR REPLACE TRIGGER audit_log_wp
  AFTER INSERT OR DELETE OR UPDATE
  ON public.well_parent
  FOR EACH ROW
  EXECUTE FUNCTION public.audit_trigger();
```

All of these triggers have been tested and work as expected.

VIEWS

The first views we will look at are associated with the web page *Table Views*.

Aquifer Abstraction - aq abstract

Total annual abstraction from transboundary aquifers

assume 12 hr/day pumping

```
CREATE OR REPLACE VIEW public.aq_abstract
 AS
```

```
SELECT ta.name AS aquifer,
   count(wp.well_no) AS "total number of wells",
   sum(h.discharge_m3_per_hr) * 180::numeric AS "annual total abstaction"
  FROM well_parent wp
    JOIN hydrogeo h ON h.well_no::text = wp.well_no::text
    JOIN trans_aquifers ta ON st_covers(ta.geom, wp.geom)
 GROUP BY ta.name;
```

Note the PostGIS function **St_covers,** which spatially links wells to transboundary aquifers. You will see this function in many of these views.

BGS hydrogeology - v_product_hydrogeo_2

Groundwater properties of BGS hydrogeological units

```
CREATE OR REPLACE VIEW public.v_product_hydrogeo_2
 AS
 SELECT wp.well_no,
   wp.orig_wellno,
   h.discharge_m3_per_hr,
   h.specific_capacity,
   h.water_strike_1_dbg,
   kse.hydrogeo_u AS "BGS Hydrogeological unit"
  FROM well_parent wp
    JOIN hydrogeo h ON wp.well_no::text = h.well_no::text
    JOIN kesomet_hydrogeo kse ON st_covers(kse.geom, wp.geom)
 ORDER BY h.discharge_m3_per_hr DESC NULLS LAST;
```

Note this query uses a shapefile kesomet-hydrogeo, that was formed by merging three shapefiles covering Kenya, Somalia, and Ethiopia. Same for the next query.

Well Hydrogeology - v_well_hydrogeo

Lithologies encountered by wells

```
CREATE OR REPLACE VIEW public.v_well_hydrogeo
 AS
 SELECT wp.well_no,
    wp.orig_wellno,
    h.lithology1_descrip AS "lithology at screen 1",
    h.lithology2_descrip AS "lithology at screen 2",
    h.lithology3_descrip AS "lithology at screen 3",
    kse.hydrogeo_u AS "BGS Hydrogeological unit",
    ta.name AS aquifer
   FROM well_parent wp
     JOIN hydrogeo h ON wp.well_no::text = h.well_no::text
     JOIN kesomet_hydrogeo kse ON st_covers(kse.geom, wp.geom)
     JOIN trans_aquifers ta ON st_covers(ta.geom, wp.geom)
  ORDER BY h.lithology1_descrip DESC NULLS LAST;
```

Water sample analysis – wq_error

Water Samples Ionic Balance

```
CREATE OR REPLACE VIEW public.wq_error
 AS
 WITH wqsums AS (
         SELECT wq.well_no,
            wq.sample_no,
            wq.ca_meql + wq.mg_meql + wq.na_meql + wq.k_meql AS sumcat,
            wq.cl_meql + wq.hco3_alk_meql + wq.so4_meql AS suman,
            wq.ca_meql + wq.mg_meql + wq.na_meql + wq.k_meql + wq.cl_meql + wq.hco3_alk_meql +
wq.so4_meql AS tdi
           FROM wq
        )
 SELECT well_no,
```

```
  sample_no,
  sumcat AS "total cations",
  suman AS "total anions",
  tdi,
  (sumcat - suman) / tdi * 100::numeric AS "percent error"
 FROM wqsums;
```

Screen analysis - v_elev_scr

Screened intervals and elevations of screens

```
CREATE OR REPLACE VIEW public.v_elev_scr
 AS
 SELECT wp.easting,
  wp.northing,
  wp.elev - c.screen1_bot_dbg AS elevation_bottom_of_first_screen,
  c.screen1_bot_dbg - c.screen1_top_dbg AS screened_interval_1,
  wp.elev - c.screen2_bot_dbg AS elevation_bottom_of_second_screen,
  c.screen2_bot_dbg - c.screen2_top_dbg AS screened_interval_2,
  wp.elev - c.screen3_bot_dbg AS elevation_bottom_of_third_screen,
  c.screen3_bot_dbg - c.screen3_top_dbg AS screened_interval_3,
  h.discharge_m3_per_hr
 FROM well_parent wp
   JOIN construction c ON c.well_no::text = wp.well_no::text
   JOIN hydrogeo h ON h.well_no::text = wp.well_no::text;
```

Well-stream distance - v_conjunctive

Well distances to nearest stream

```
CREATE OR REPLACE VIEW public.v_conjunctive
 AS
 SELECT wp.well_no,
```

```
   wp.orig_wellno,
   st_distance(wp.geom::geography, ksm.geom::geography) / 1000::numeric::double precision AS
"nearest distance to stream km",
   ta.name AS aquifer,
   ksm.catch_skm AS "catchment area",
   ta.annual_rainfall_mm - ta.annual_et AS "annual excess precipitation mm",
   ((ta.annual_rainfall_mm - ta.annual_et) * 1000)::numeric::double precision * ksm.catch_skm AS
"annual excess precipitation cubic meters"
  FROM well_parent wp
    JOIN trans_aquifers ta ON st_covers(ta.geom::geography, wp.geom::geography)
    JOIN kesomet_rivers ksm ON st_dwithin(ksm.geom::geography, wp.geom::geography,
5000::double precision)
  ORDER BY (st_distance(wp.geom::geography, ksm.geom::geography) / 1000::numeric::double
precision);
```

kesomet_rivers is also a shape file, this one comprising river networks in Kenya, Somalia, and Ethiopia. Note the spatial functions **st_dwithin** and **st_covers**

water strike elevations – v_ws_elev

Elevations of water strikes

```
CREATE OR REPLACE VIEW public.v_ws_elev
 AS
 SELECT wp.easting,
   wp.northing,
   wp.elev - h.water_strike_1_dbg AS elevation_first_water_strike,
   wp.elev - h.water_strike_2_dbg AS elevation_second_water_strike,
   h.discharge_m3_per_hr
  FROM well_parent wp
    JOIN hydrogeo h ON h.well_no::text = wp.well_no::text;
```

There other views which are used for pages in the *graphs* and *map* modules. They are summarized below.

The following two views use the elevation values in the well_parent table to calculate elevations of other levels (screen elevations, depths, water strikes etc) in the construction and hydrogeo tables

```
CREATE OR REPLACE VIEW public.construction_elevation
 AS
 SELECT c.well_no,
    wp.elev,
    c.depth_dbg,
    wp.elev - c.depth_dbg AS "elevation well bottom",
    c.screen1_top_dbg AS "screen 1 top",
    wp.elev - c.screen1_top_dbg AS "elevation screen 1 top",
    c.screen1_bot_dbg AS "screen 1 bottom",
    wp.elev - c.screen1_bot_dbg AS "elevation screen 1 bottom",
    c.screen2_top_dbg AS "screen 2 top",
    wp.elev - c.screen2_top_dbg AS "elevation screen 2 top",
    c.screen2_bot_dbg AS "screen 2 bottom",
    wp.elev - c.screen2_bot_dbg AS "elevation screen 2 bottom",
    c.screen3_top_dbg AS "screen 3 top",
    wp.elev - c.screen3_top_dbg AS "elevation screen 3 top",
    c.screen3_bot_dbg AS "screen 3 bottom",
    wp.elev - c.screen3_bot_dbg AS "elevation screen 3 bottom"
   FROM well_parent wp
     JOIN construction c ON c.well_no::text = wp.well_no::text;

ALTER TABLE public.construction_elevation
    OWNER TO postgres;
```

```
COMMENT ON VIEW public.construction_elevation
   IS 'calculates elevation of all screens in construction table from the elevation in parent well table';


CREATE OR REPLACE VIEW public.hydrogeo_elevation
 AS
 SELECT h.well_no,
   wp.elev,
   h.swl_dbg AS swl,
   wp.elev - h.swl_dbg AS "elevation swl",
   h.dwl_dbg AS dwl,
   wp.elev - h.dwl_dbg AS "elevation dwl",
   h.depth_br_bg AS "depth to bedrock",
   wp.elev - h.depth_br_bg AS "elevation of bedrock",
   h.water_strike_1_dbg AS "first water strike",
   wp.elev - h.water_strike_1_dbg AS "elevation first strike",
   h.water_strike_2_dbg AS "second water strike",
   wp.elev - h.water_strike_2_dbg AS "elevation second water strike"
  FROM well_parent wp
    JOIN hydrogeo h ON h.well_no::text = wp.well_no::text;
```

Another view was used to make interpolation of variables in the web application easier to code

```
CREATE OR REPLACE VIEW public.interp1
 AS
 SELECT wp.well_no,
   wp.orig_wellno,
   wp.geom,
   c.depth_dbg,
```

```
  h.water_strike_1_dbg,
  h.discharge_m3_per_hr,
  h.transmissivity,
  h.storage_coefficient,
  wq.ca_meql,
  wq.mg_meql,
  wq.na_meql,
  wq.k_meql,
  wq.cl_meql,
  wq.so4_meql
 FROM well_parent wp
   LEFT JOIN construction c ON c.well_no::text = wp.well_no::text
   LEFT JOIN hydrogeo h ON h.well_no::text = wp.well_no::text
   LEFT JOIN wq ON wq.well_no::text = wp.well_no::text;
```

Indexes

There are numerous indexes in the database on columns that are presumed to be used often in query searches, Notably, these are columns that are frequently referred to in the WHERE clause.

There is a tradeoff between the speed of searches and the speed of inserting or upating tables because the index, which in itself is a table, needs to be rebuilt every time records are changed in the table it is applied to.

For example, the crs_lookup table has the index **crsIndex** on the **srid** column**.** This was created because the trigger function **fn_make_pointgeom** needs to find the coordinate system the user enters in the well_parent form to convert text values to geometry. Since there are over 5000 coordinate systems in the spatial_ref_system table with a primary key of srid (which automatically has an index) the lookup is quite fast and time consuming sequential scans are avoided (sequential scans are covered in the ANALYZE section). As we don't expect records in the crs_lookup table to be updated or inserted very often, this is an ideal choice

The syntax to create an index:

```
CREATE INDEX IF NOT EXISTS "crsIndex"
  ON public.crs_lookup USING btree
  (srid ASC NULLS LAST)
  WITH (deduplicate_items=True)
  TABLESPACE pg_default;


COMMENT ON INDEX public."crsIndex"
  IS 'To speed up looking up coordinate systems in the function fn_make_pointgeom';
```

A full list of indexes is provided in the Data Dictionary in the Annex.

Constraints

There are four major types of contstaints:

1. Primary Key
2. Foreign Key
3. Unique
4. Check

A primary key is a unique identifier and almost every table has one. A foreign key links the column of one table with a duplicate column in another table to allow joining tables. A unique constraint insures a unique value in that column for every record (primary keys are unique by default). A check constraint evaluates if a value entered is valid through an expression (e.g. start_date < end_date).

In the GWIS database, all constraints except Check are used – valid inputs are checked at the client end to avoid the database throwing an error back at the browser (which will confuse the user).

A full list of constraints is provided in the Data Dictionary in the Annex.

ACCESSING THE GWIS DATABASE

The database can be accessed with a command line interface (CLI) called psql.

To connect type: psql -h 197.254.1.11 -p 5432 -U postgres

-h means host so type in the Ip address, -p is the port, which is 5432, and the user is postgres, a superuser.

You will get this prompt : postgres=# This means you are connected to the default postgres database.

Type postgres=# SHOW config_file**;** (note the semi-colon at the end)

config_file

/etc/postgresql/16/main/postgresql.conf

(1 row)

This is what we saw before while exploring directories

Same for pg_hba.conf :

postgres=# show hba_file**;**

hba_file

/etc/postgresql/16/main/pg_hba.conf

(1 row)

Remember these locations because they will be needed when migrating to a new server or reinstalling (details covered later in this section).

Type show data_directory;

data_directory

/var/lib/postgresql/16/main

(1 row)

Now Type \c gwis to connect to the gwis database:

```
postgres=# \c gwis
psql (16.1, server 16.4 (Ubuntu 16.4-0ubuntu0.24.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
You are now connected to database "gwis" as user "postgres".
gwis=#
```

Notice the prompt changes to gwis – that's the name of the database.

Type \l

```
gwis=# \l
                                                List of databases
   Name    |  Owner   | Encoding | Locale Provider |   Collate   |    Ctype    | ICU Locale | ICU Rules |   Access privileges
-----------+----------+----------+-----------------+-------------+-------------+------------+-----------+-----------------------
 gwis      | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           |
 postgres  | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           |
 template0 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           | =c/postgres          +
           |          |          |                 |             |             |            |           | postgres=CTc/postgres
 template1 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           | =c/postgres          +
           |          |          |                 |             |             |            |           | postgres=CTc/postgres
(4 rows)
```

This shows all the databases in the system.

Type \d

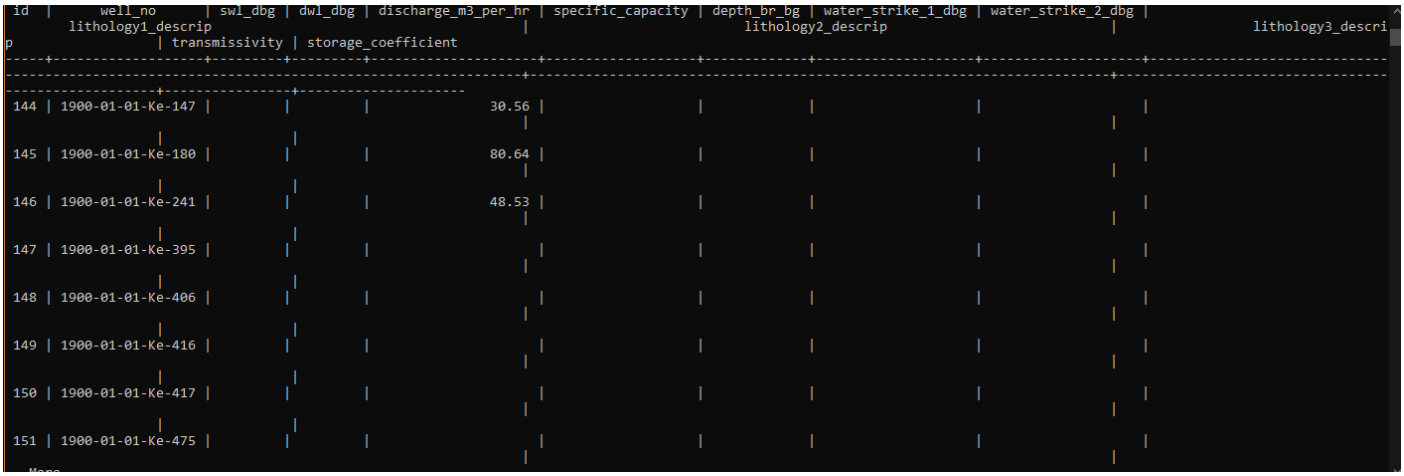
This shows all the database objects in the gwis database . Press spacebar to see more.

```
                 List of relations
 Schema |            Name            |   Type   |  Owner
--------+----------------------------+----------+----------
 public | Geo_fsm_zone1_id_seq       | sequence | postgres
 public | aq_abstract                | view     | postgres
 public | construction               | table    | postgres
 public | construction_elev          | view     | postgres
 public | crs_lookup                 | table    | postgres
 public | crs_lookup_rec_id_seq      | sequence | postgres
 public | drilling_meth              | table    | postgres
 public | drilling_meth_id_seq       | sequence | postgres
 public | garissa_ke_annual_total    | view     | postgres
 public | garissa_ke_monthly_avg     | view     | postgres
 public | garissa_ke_rain            | table    | postgres
 public | geo_fsm_zone1              | table    | postgres
 public | geo_fsm_zone2              | table    | postgres
 public | geo_fsm_zone2_id_seq       | sequence | postgres
 public | geo_fsm_zone3              | table    | postgres
 public | geo_fsm_zone3_id_seq       | sequence | postgres
 public | geo_fsm_zone4              | table    | postgres
 public | geo_fsm_zone4_id_seq       | sequence | postgres
 public | geo_fsm_zone5              | table    | postgres
 public | geo_fsm_zone5_id_seq       | sequence | postgres
 public | geo_fsm_zone6              | table    | postgres
 public | geo_fsm_zone6_id_seq       | sequence | postgres
 public | geography_columns          | view     | postgres
 public | geometry_columns           | view     | postgres
 public | hydrogeo                   | table    | postgres
 public | hydrogeo_elevation         | view     | postgres
-- More  --
```

Type \d hydrogeo - You will get a list of the columns in the table hydrogeo and their properties.

```
gwis=# \d hydrogeo
                                          Table "public.hydrogeo"
      Column       |          Type          | Collation | Nullable |                                  Default
-------------------+------------------------+-----------+----------+-------------------------------------------------------------------------
 id                | integer                |           | not null | nextval('hydrogeo_id_seq'::regclass)
 well_no           | character varying(100) |           | not null |
 swl_dbg           | numeric(6,2)           |           |          |
 dwl_dbg           | numeric(6,2)           |           |          |
 discharge_m3_per_hr | numeric(6,2)         |           |          |
 specific_capacity | numeric(6,2)           |           |          | generated always as ((discharge_m3_per_hr / (dwl_dbg - swl_dbg))) stored
 depth_br_bg       | numeric(6,2)           |           |          |
 water_strike_1_dbg | numeric(6,2)          |           |          |
 water_strike_2_dbg | numeric(6,2)          |           |          |
 lithology1_descrip | character varying(100) |           |          |
 lithology2_descrip | character varying(100) |           |          |
 lithology3_descrip | character varying(100) |           |          |
 transmissivity    | numeric(6,2)           |           |          |
 storage_coefficient | numeric(6,6)          |           |          |
Indexes:
    "hydro_key" PRIMARY KEY, btree (well_no)
    "idx_swl_dbg" btree (swl_dbg) INCLUDE (swl_dbg) WITH (deduplicate_items='true')
Foreign-key constraints:
    "well_no" FOREIGN KEY (well_no) REFERENCES well_parent(well_no) ON UPDATE CASCADE ON DELETE CASCADE
```

SELECT * FROM hydrogeo;

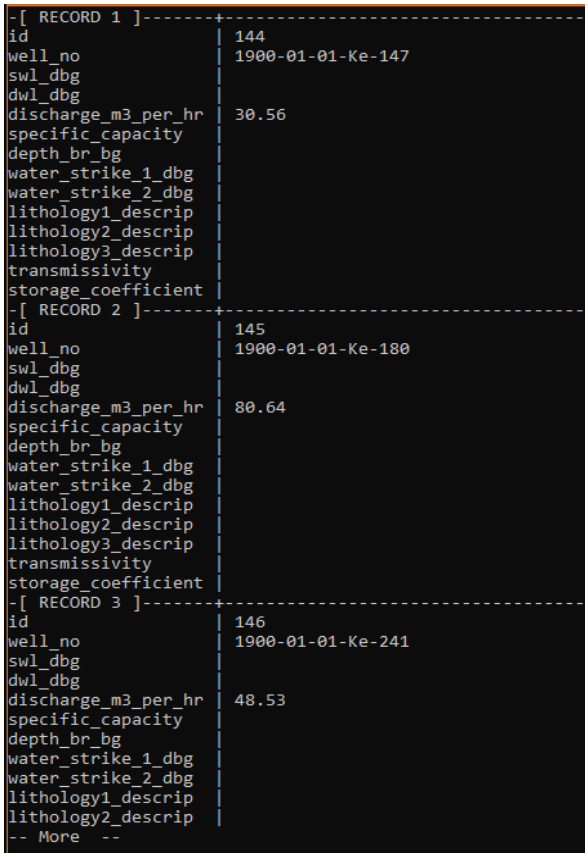


This is hard to read. Press the spacebar until all the records are shown and you'll get the prompt back. Then type \x

```
gwis=# \x
Expanded display is on.
gwis=#
```

Now press the up-arrow key 2 times to get the

SELECT statement back



This is much more readable.

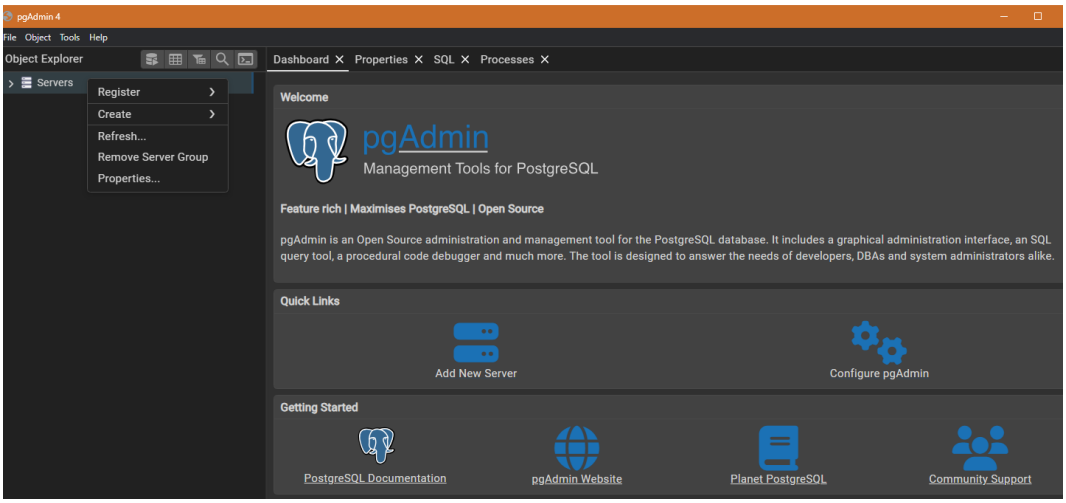
Type \q to get out of reading the rest of the records

The command line is very valuable. Most of the time, however, you will use a GUI interface such as pgAdmin, DBeaver, or phpPgAdmin, free third party downloads, but for certain tasks, like the COPY command and for backup and restore, the command line is the preferred way.

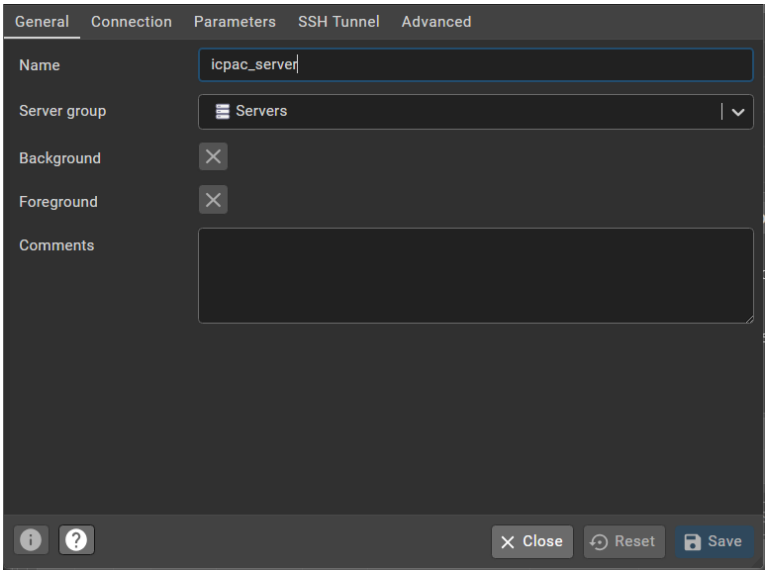
The GUI interface demonstrated in the manual is pgAdmin.

Introduction to the pgAdmin interface

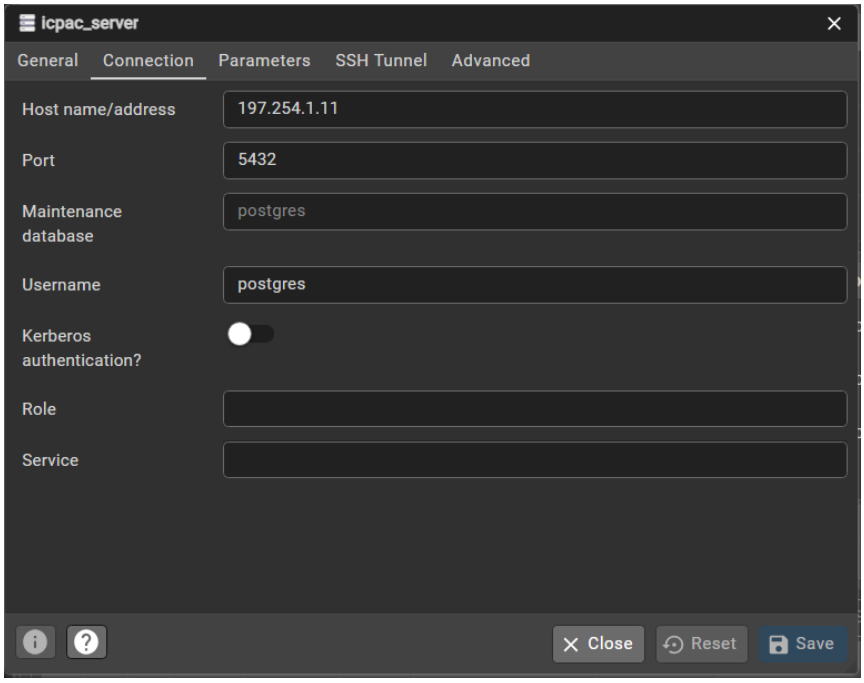
Opening screen



Choose Register, then Server



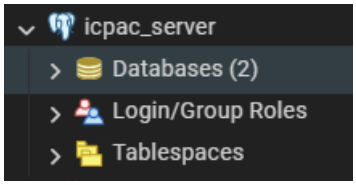
Give a name such as icpac_server, or any name you like, then click Connection

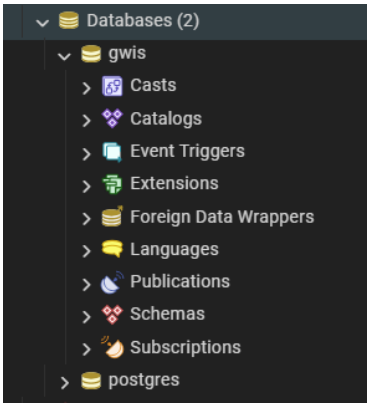


Fill in the information as shown. For now, don't bother with anything else. Click Save

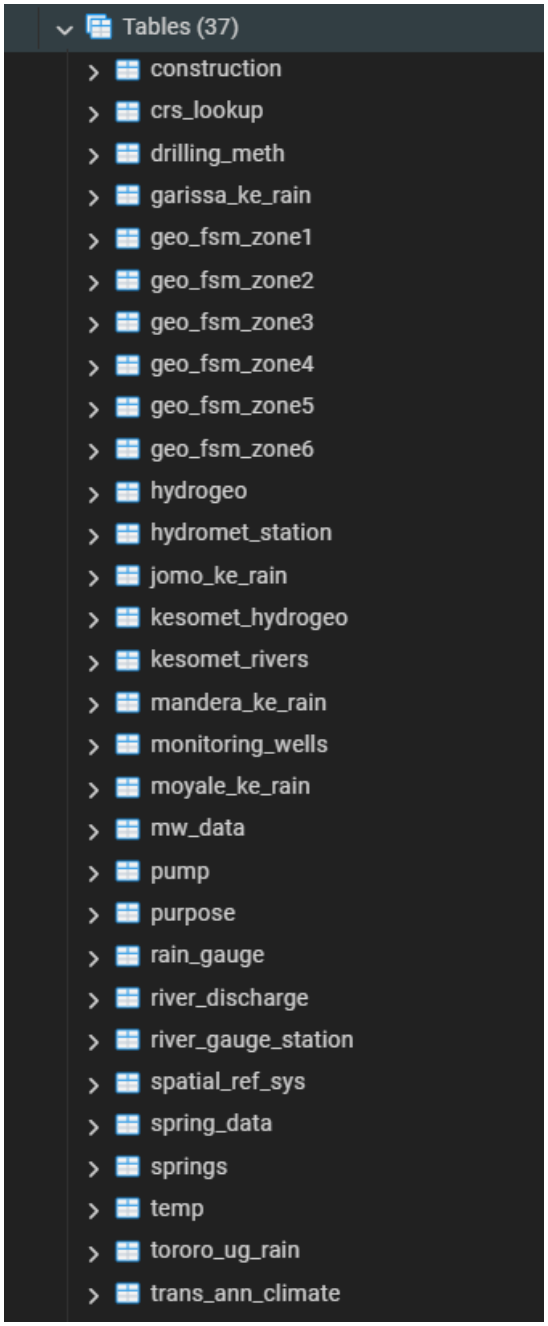
The server name should appear in the left pane. Click on the > symbol

There are 2 databases – one is the default postgres database, the other is gwis which the consultant created





Click on the > next to Schemas and then > next to public and then > nets to table, to get a list of tables n the gwis database.



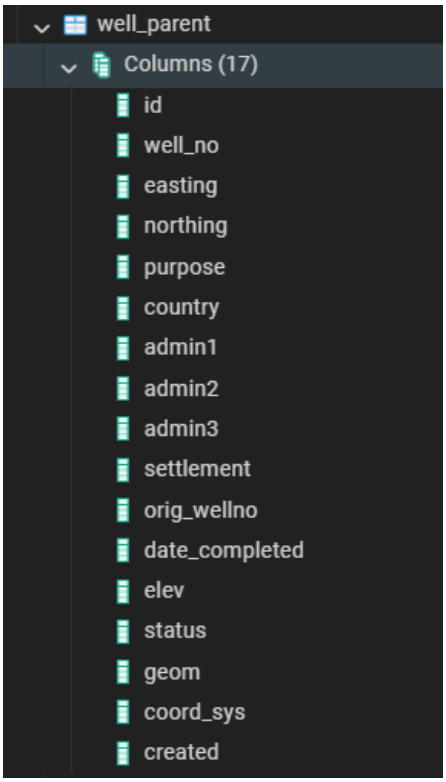
Scroll down to the well_parent table and click its arrow symbol

You will see a list of the types of objects associated with the table.

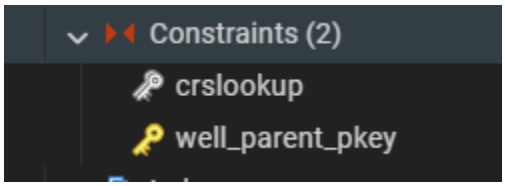
Expand the tree to see details of all the types

right click for the context menu

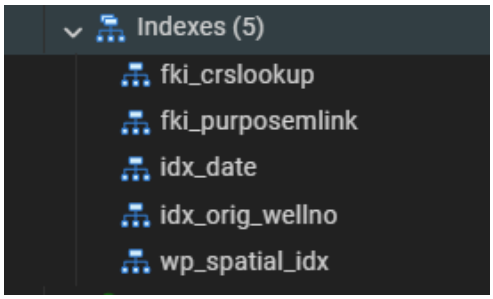
Columns



Constraints

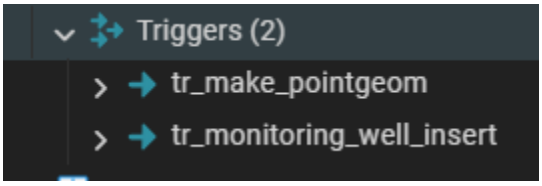


Indexes

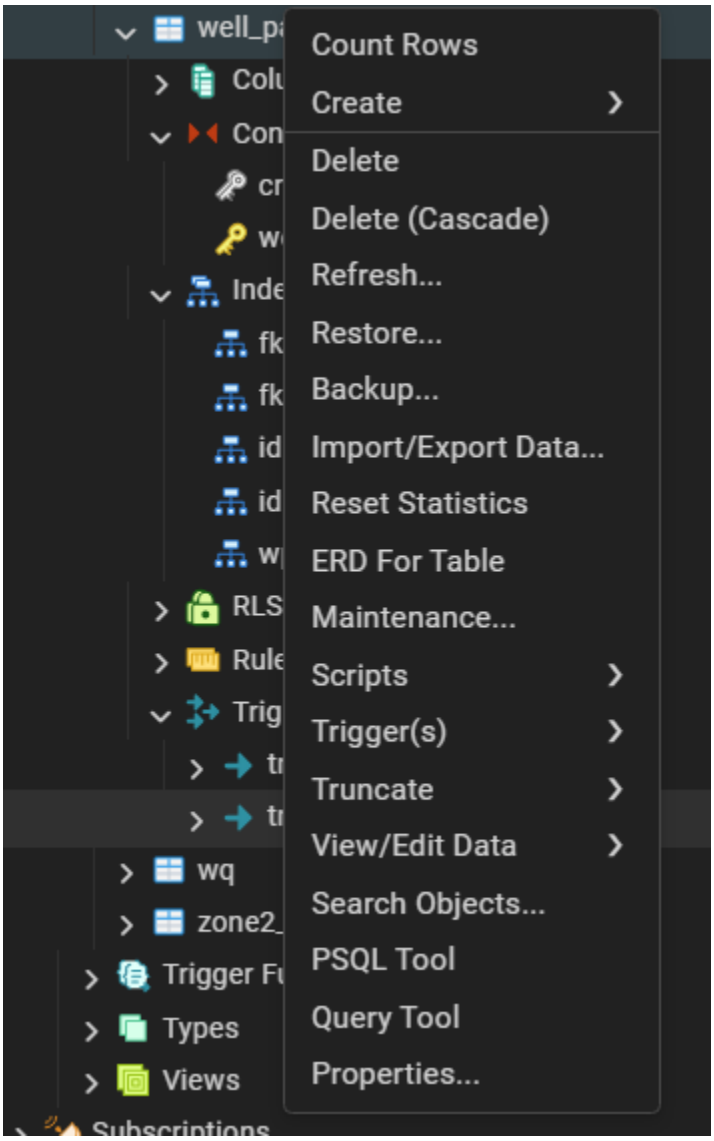


For constraints we have a primary key and a foreign key to the crs_lookup table. For indexes we have automatically-generated indexes on foreign keys and the primary key, plus 2 keys created by the consultant, one on date completed and the other is a spatial index needed to speed up spatial queries.

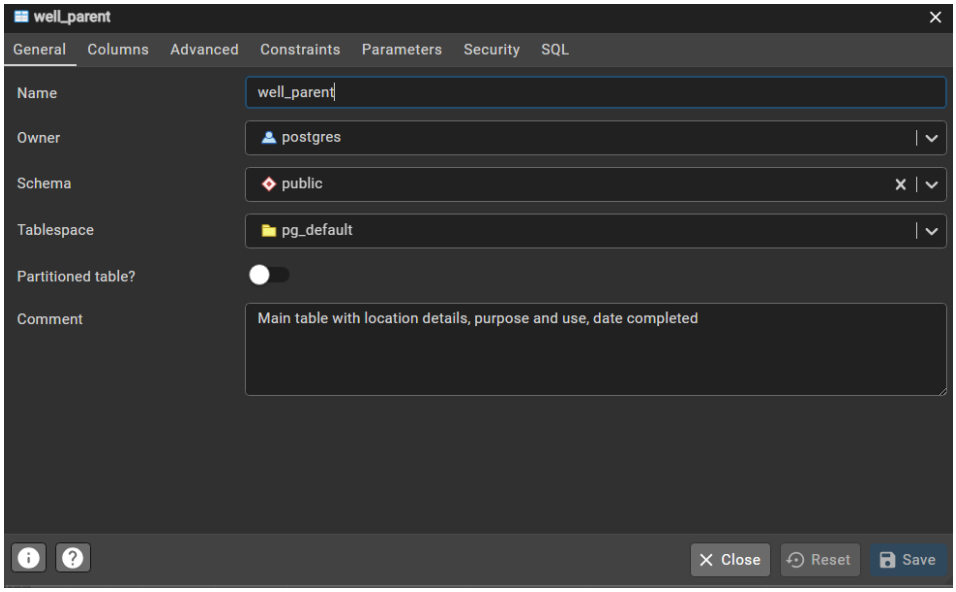
We also have 2 triggers mentioned earlier in this document.



If you right click on any object you can view its properties. Go up on level to the well_parent table and right-click

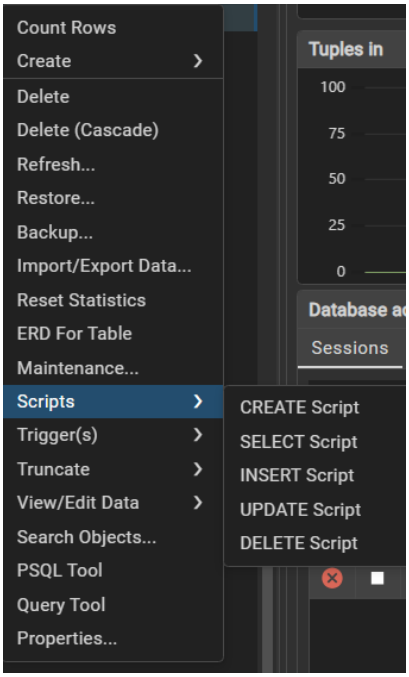


Scroll to the bottom and click properties



Click on the various headings to see the information.

There are two other options in the context menu we will mention. If you go to scripts, and then click CREATE Script



The main panel, the Query panel, displays the SQL script that creates the table

```
CREATE TABLE IF NOT EXISTS public.well_parent
(
    id integer NOT NULL DEFAULT nextval('well_parent_id_seq'::regclass),
    well_no character varying(50) COLLATE pg_catalog."default" NOT NULL,
    easting numeric(14,6),
    northing numeric(14,6),
    purpose character varying(25) COLLATE pg_catalog."default" NOT NULL,
    country character varying(50) COLLATE pg_catalog."default" NOT NULL,
    admin1 character varying(50) COLLATE pg_catalog."default",
    admin2 character varying(50) COLLATE pg_catalog."default",
    admin3 character varying(50) COLLATE pg_catalog."default",
    settlement character varying(250) COLLATE pg_catalog."default",
    orig_wellno character varying(50) COLLATE pg_catalog."default" NOT NULL,
    date_completed date NOT NULL,
    elev numeric(7,2),
    status boolean,
    geom geometry(Point,4326),
    coord_sys integer,
    created timestamp without time zone DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT well_parent_pkey PRIMARY KEY (well_no),
    CONSTRAINT crslookup FOREIGN KEY (coord_sys)
        REFERENCES public.crs_lookup (srid) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        DEFERRABLE INITIALLY DEFERRED
)

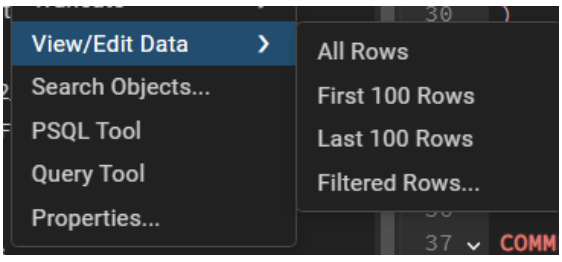
TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.well_parent
    OWNER to postgres;

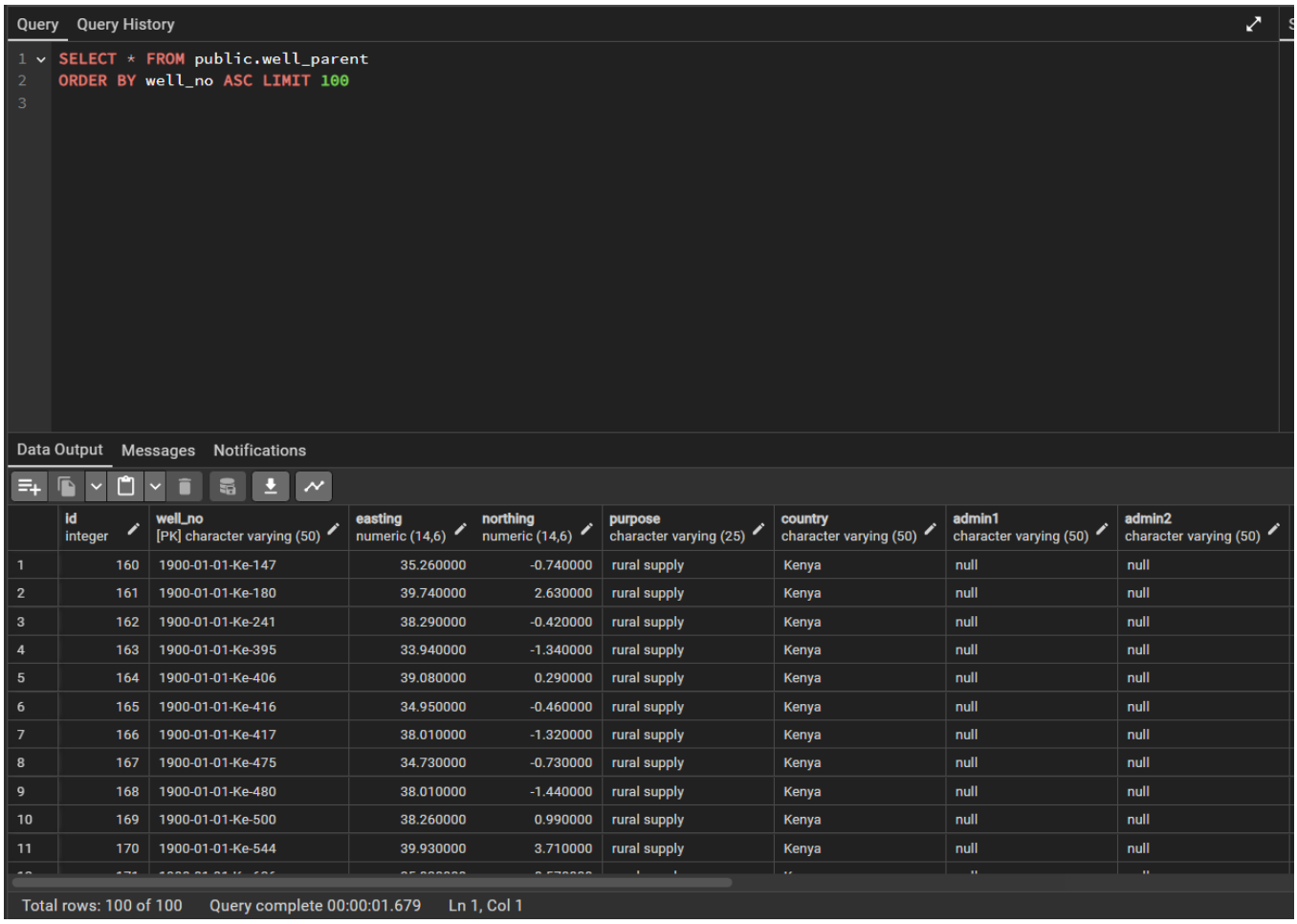
COMMENT ON TABLE public.well_parent
    IS 'Main table with location details, purpose and use, date completed';

COMMENT ON COLUMN public.well_parent.well_no
    IS 'This is the primary key of the table and the foreign key for all child tables';
```

Another option in the context menu is View/Edit data



Click on First 100 rows so we don't 'have to wait so long for the results



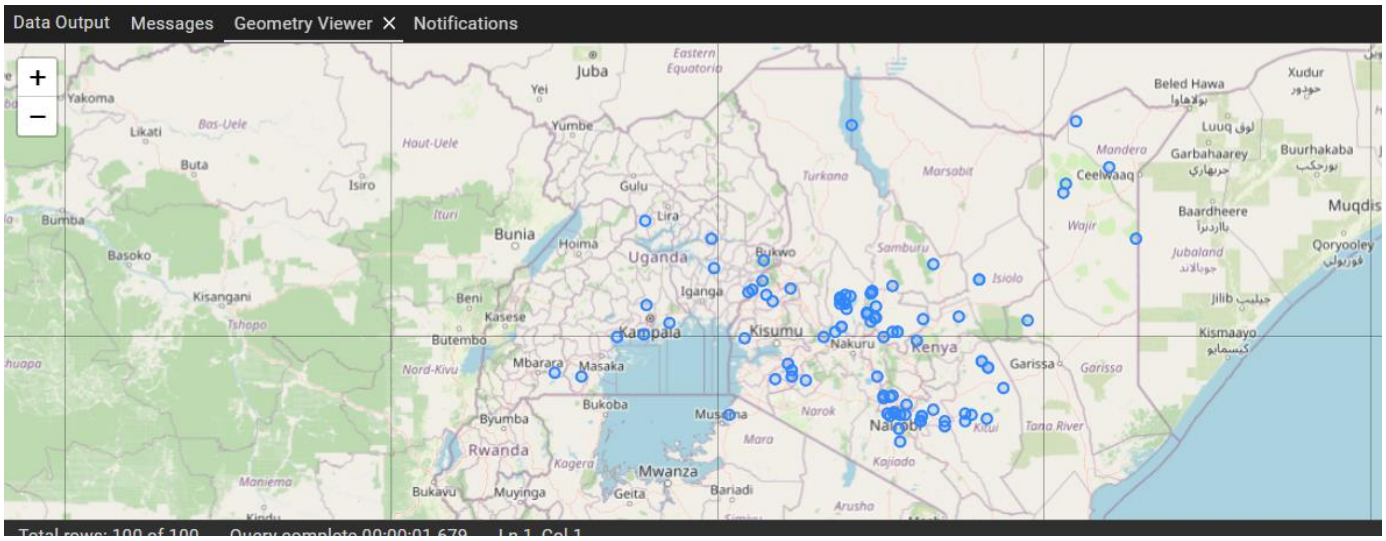
	id integer	well_no [PK] character varying (50)	easting numeric (14,6)	northing numeric (14,6)	purpose character varying (25)	country character varying (50)	admin1 character varying (50)	admin2 character varying (50)
1	160	1900-01-01-Ke-147	35.260000	-0.740000	rural supply	Kenya	null	null
2	161	1900-01-01-Ke-180	39.740000	2.630000	rural supply	Kenya	null	null
3	162	1900-01-01-Ke-241	38.290000	-0.420000	rural supply	Kenya	null	null
4	163	1900-01-01-Ke-395	33.940000	-1.340000	rural supply	Kenya	null	null
5	164	1900-01-01-Ke-406	39.080000	0.290000	rural supply	Kenya	null	null
6	165	1900-01-01-Ke-416	34.950000	-0.460000	rural supply	Kenya	null	null
7	166	1900-01-01-Ke-417	38.010000	-1.320000	rural supply	Kenya	null	null
8	167	1900-01-01-Ke-475	34.730000	-0.730000	rural supply	Kenya	null	null
9	168	1900-01-01-Ke-480	38.010000	-1.440000	rural supply	Kenya	null	null
10	169	1900-01-01-Ke-500	38.260000	0.990000	rural supply	Kenya	null	null
11	170	1900-01-01-Ke-544	39.930000	3.710000	rural supply	Kenya	null	null

On the top we have the SQL query that was executed (‘ * ’ means all fields) and the bottom part shows the results, including the column names and datatype, and at the bottom of that we have the number of rows returned and the time it took – in this case 1.679 seconds.

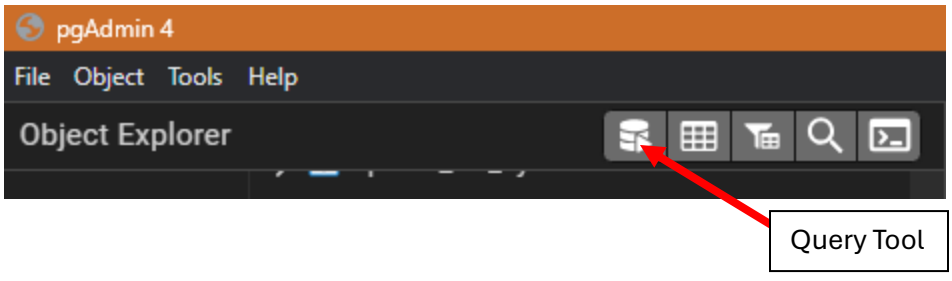
Scroll to the right to see the last 2 columns – one is geometry and the other is created. Created records the date and time this record was entered

The geom column of type geometry has an icon in the column header. Click on that and a map is displayed showing where those records (wells) are located.

orig_wellno character varying (50)	date_completed date	elev numeric (7,2)	status boolean	geom geometry	coord_sys integer	created timestamp without time zone
Ke072	1900-01-01	1880.00	[null]	0101000020E6100000E17A14AE47A14140AE47E17A14AEE7BF	4326	2024-05-08 00:00:00
Ke071	1900-01-01	496.00	[null]	0101000020E61000001F85EB51B8DE43400AD7A3703D0A05...	4326	2024-05-08 00:00:00
Ke063	1900-01-01	787.00	[null]	0101000020E610000085EB51B81E254340E17A14AE47E1DABF	4326	2024-05-08 00:00:00
Ke033	1900-01-01	1134.00	[null]	0101000020E6100000B81E85EB51F84040713D0AD7A370F5BF	4326	2024-05-08 00:00:00
Ke025	1900-01-01	329.00	[null]	0101000020E61000000AD7A3703D8A43408FC2F5285C8FD2...	4326	2024-05-08 00:00:00
Ke055	1900-01-01	1589.00	[null]	0101000020E61000009A9999999794140713D0AD7A370DD...	4326	2024-05-08 00:00:00
Ke068	1900-01-01	1183.00	[null]	0101000020E6100000E17A14AE470143401F85EB51B81EF5BF	4326	2024-05-08 00:00:00
Ke056	1900-01-01	1716.00	[null]	0101000020E61000003D0AD7A3705D41405C8FC2F5285CE7...	4326	2024-05-08 00:00:00
Ke070	1900-01-01	1064.00	[null]	0101000020E6100000E17A14AE470143400AD7A3703D0AF7...	4326	2024-05-08 00:00:00
Ke066	1900-01-01	518.00	[null]	0101000020E6100000E17A14AE47214340AE47E17A14AEEF3F	4326	2024-05-08 00:00:00
Ke061	1900-01-01	754.00	[null]	0101000020E6100000D7A3703D0AF74340AE47E17A14AE0D...	4326	2024-05-08 00:00:00



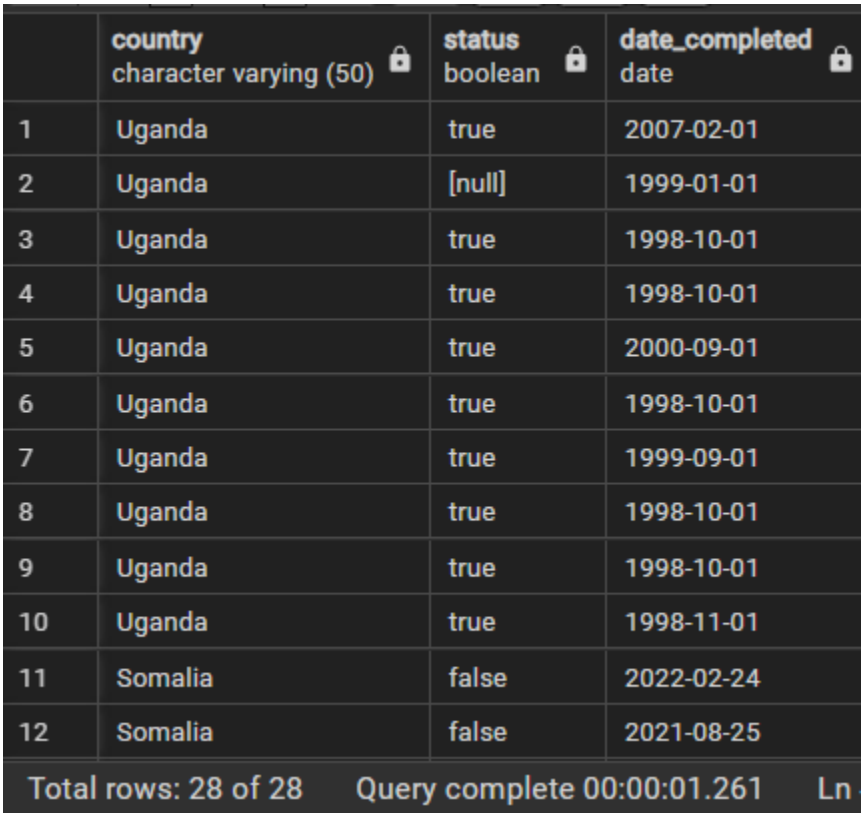
You can use execute your own queries by using the Query Builder



Click on it to get a new query window and type this

```
SELECT country, status, date_completed
FROM well_parent
WHERE purpose = 'monitoring'
order by country desc
```

here are the results:



	country character varying (50)	status boolean	date_completed date
1	Uganda	true	2007-02-01
2	Uganda	[null]	1999-01-01
3	Uganda	true	1998-10-01
4	Uganda	true	1998-10-01
5	Uganda	true	2000-09-01
6	Uganda	true	1998-10-01
7	Uganda	true	1999-09-01
8	Uganda	true	1998-10-01
9	Uganda	true	1998-10-01
10	Uganda	true	1998-11-01
11	Somalia	false	2022-02-24
12	Somalia	false	2021-08-25

Total rows: 28 of 28 Query complete 00:00:01.261 Ln

During the training, the pgAdmin client interface will be thoroughly examined.

DATABASE ADMINISTRATION

The topic of database administration could make up a year-long course. Only basic essential tasks are covered in this manual. It is expected that IGAD/ICPAC will have in-house staff to monitor and maintain the database.

BACKUP & RESTORE

There are numerous ways to backup and restore a database in PostgreSQL. This manual covers only one method. For alternative approaches to backing up and restoring databases, you can refer to the PostgreSQL documentation. The possibility of using pgAgent of pg_cron to schedule backups will be discussed during training.

There are two different executables involved:

1. **pg_dump** is used to create the backup script (written in SQL)
2. **psql,** the familiar CLI introduced earlier, is used to restore that file.

Dumps can be output in script or archive file formats. Script dumps are plain-text files containing the SQL commands required to reconstruct the database to the state it was in at the time it was saved. This is what we will use. To restore from such a script, feed it to psql. Script files can be used to reconstruct the database even on other machines and other architectures; with some modifications, even on other SQL database products.

To dump a database into an SQL script, first decide what location you're going to put it in. In this example we are backing up the gwis database on the server to a script file called Sept1_bak.sql and putting it a directory on our local machine C drive in a directory called testbackup:

pg_dump -h 197.254.1.11 -U postgres -d gwis >C:\testbackup\aug22C_bak.sql

Note the greater than symbol '>'

We could have saved the file on the server, but it's best to keep it on a different machine.

To restore the file, we first need to create the database that all the objects will be put into. It is possible to put the create database command into the sci-t but typically this is not done.

To create the database use this script:

```
CREATE DATABASE gwis
  WITH
  OWNER = postgres
  ENCODING = 'UTF8'
  LOCALE_PROVIDER = 'libc'
  CONNECTION LIMIT = -1
  IS_TEMPLATE = False;
```

To populate the database with the savd objects, use psql as in the following command:

psql -U postgres -h 197.254.1.11 -d gwis < C:\testbackup\aug22C_bak.sql

Note the less than symbol '<'

In place of these symbols, you could type -f

pg_dump -h 197.254.1.11 -U postgres -d gwis -f C:\testbackup\aug22C_bak.sql

psql -U postgres -h 197.254.1.11 -d gwis < -f C:\testbackup\aug22C_bak.sql

Also keep in mind, when referring to the file in these commands, the full path is required.

COPY COMMAND

The COPY command is used to copy data between a file and a table, it appends records to those already present. The file format is usually a csv file. COPY TO copies the contents of a table to a file, while COPY FROM copies data from a file to a table.

If a column list is specified, COPY TO copies only the data in the specified columns to the file. For COPY FROM, each field in the file is inserted, in order, into the specified column. Table columns not specified in the COPY FROM column list will receive their default values.

Standard syntax:

```
COPY table_name [ ( column_name [, ...] ) ]
  FROM { 'filename' | PROGRAM 'command' | STDIN }
  [ [ WITH ] ( option [, ...] ) ]
  [ WHERE condition ]
```

COPY { table_name [(column_name [, ...])] | (query) }

TO { 'filename' | PROGRAM 'command' | STDOUT }

[[WITH] (option [, ...])]

Example 1 with error

COPY well_parent (well_no,easting,northing,purpose,country,admin1,admin2,admin3,settlement,

orig_wellno,date_completed,elev,status, coord_sys)

FROM 'D:\IGAD PROJECT\csv for import\Aug_Ke\well_parent.csv'

WITH (FORMAT CSV,HEADER,DELIMITER ',')

ERROR: date/time field value out of range: "22/07/2017"

HINT: Perhaps you need a different "datestyle" setting.

We got this error because the date format should be year-month-day -> 2017-07-22

So be sure, if you're using Excel, to format the date in that manner.

Example 2

COPY hydrogeo (well_no, swl_dbg, dwl_dbg, discharge_m3_per_hr,water_strike_1_dbg,

water_strike_2_dbg, lithology1_descrip, lithology2_descrip, lithology3_descrip, transmissivity, storage_coefficient)

FROM '/var/lib/postgresql/16/hydrogeo.csv'

WITH (FORMAT CSV,HEADER,DELIMITER ',');

Other mistakes:

make sure not to include id in columns to insert - same for created – as these are automatically.

generated. Also for geom column, as these values are created by the trigger function mentioned earlier.

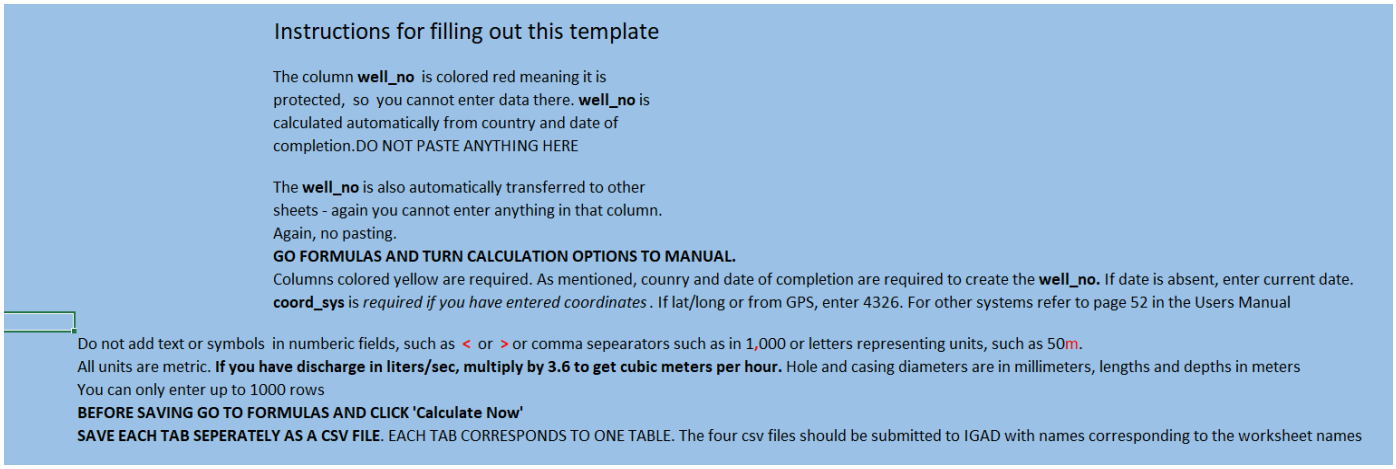
If you use the template provided, these mistakes can be avoided.

Also note that the constraints disallowing negative numbers, text or symbols in numeric fields, and chemical parameter ranges that were implemented on the client side are also implemented server-side inside the database. Data validation cannot be implemented in Excel when filling in the template because copyng and pasting into the cells overwrites the validation rules. (see next topic on Excel templates). So if such dirty data is in the csv files you are trying to bring into the table, PostgreSQL will throw an error and the COPY command will fail. This is the responsibility of the data processing person.

EXCEL TEMPLATES TO CREATE CSV FILES

Download the Excel templates from the Knowledge Base – Resources for Administrators page.

The first tab has instructions which should be read and made familiar.



Instructions for filling out this template

The column **well_no** is colored red meaning it is protected, so you cannot enter data there. **well_no** is calculated automatically from country and date of completion.DO NOT PASTE ANYTHING HERE

The **well_no** is also automatically transferred to other sheets - again you cannot enter anything in that column. Again, no pasting.

GO FORMULAS AND TURN CALCULATION OPTIONS TO MANUAL.

Columns colored yellow are required. As mentioned, counry and date of completion are required to create the **well_no.** If date is absent, enter current date.

coord_sys is *required if you have entered coordinates* . If lat/long or from GPS, enter 4326. For other systems refer to page 52 in the Users Manual

Do not add text or symbols in numberic fields, such as < or > or comma sepearators such as in 1,000 or letters representing units, such as 50m.

All units are metric. **If you have discharge in liters/sec, multiply by 3.6 to get cubic meters per hour.** Hole and casing diameters are in millimeters, lengths and depths in meters

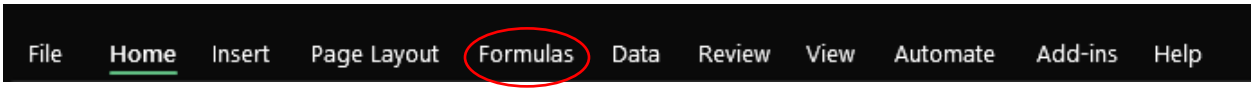
You can only enter up to 1000 rows

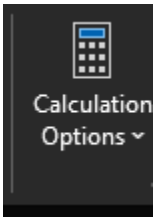
BEFORE SAVING GO TO FORMULAS AND CLICK 'Calculate Now'

SAVE EACH TAB SEPERATELY AS A CSV FILE. EACH TAB CORRESPONDS TO ONE TABLE. The four csv files should be submitted to IGAD with names corresponding to the worksheet names

IMPORTANT – before entering data

GO TO FORMULAS AND TURN CALCULATION OPTIONS TO **MANUAL.**





The second worksheet has information for the well_parent table

A	B	C	D	E	F	G	H	I	J	K	L	M	N
well_no	easting	northing	purpose	country	admin1	admin2	admin3	settlement	orig _wellno	date_completed	elev	status	coord_sys
1900-01-00--215													
1900-01-00--312													
1900-01-00--286													
1900-01-00--677													
1900-01-00--343													
1900-01-00--332													
1900-01-00--915													
1900-01-00--350													

Columns highlighted in yellow are required. The column **coord_sys** in lighter color is **required if coordinates are entered**. This the EPSG code. For GPS devices the default is WGS 84, which is code 4326. Refer to the table in the User Manual page 52 for other systems.

The other worksheets are straightforward. Note that sample_no in the Water Chemistry worksheet is required.

IMPORTANT

BEFORE SAVING GO TO FORMULAS AND CLICK 'Calculate Now'

SAVE EACH SHEET SEPERATELY AS A CSV FILE. EACH SHEET CORRESPONDS TO ONE TABLE. The four CSV files should be submitted to IGAD with names corresponding to the worksheet names

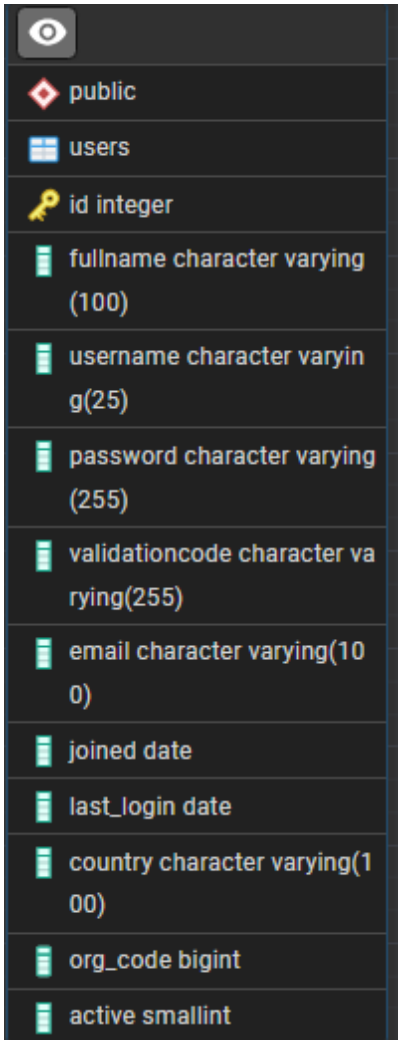
DELETING RECORDS

The database administrator is the one who can delete records, not registered editors (although in the prototype version of the GWIS this was enabled). Caution is advised and two important aspects are considered here. The DELETE command must have a filter via the WHERE clause, because without it, you will delete every record in the table!!!

DELETE FROM well_parent **WHERE** orig_wellno = bh123

The other aspect is that the foreign keys in the other tables have actions associated with them termed ON DELETE CASCADE. **This means if you delete a record in the well_parent table, you will delete the records for that well number in the other three well tables as well** (construction hydrogeo, wq).

REGISTERED EDITORS AND THE USERS TABLE



Herein is the structure of the users table. It was set up with a previous, more sophisticated login/registrations system that has since been dismantled. At the request of the Project director.

If a user is not in this table, they won't be able to log in.

The columns that remain important are fullname, username, password, email, country, and active. This last column is of Boolean type (1 or 0, true of false) and is a method for deactivating a pre-existing user.

Country is an important field, because when a users logs in, their data is retrieved from the database and kept in session variables in the application code. This is the mechanism whereby users can only insert/update records for the country they belong to.

Validation code, no longer used, was for email verification.

Because the previous registration system no longer functions**, it is the duty of the database administrator to place the user information into the table using insert and update SQL commands**. This would be done after acceptance of that user and associated organization to have the privilege of editing records.

CONFIGURATION FILES

There two configuration files of concern: pg_hba.conf and postgresql.conf, both located in

/etc/postgresql/16/main. At the moment these files are in the correct configuration, but in case of migration or re-installation they will need to be edited for the application to work and to access the database through clients such as pgAdmin or QGIS.

pg_hba.conf

Client authentication is controlled by pg_hba.conf (HBA stands for host-based authentication). The pg_hba.conf file defines a variety of aspects:

- What IP addresses or ranges can connect to PostgreSQL and for what accounts
- What method of authentication those IP ranges / accounts can use

Change to user postgres, navigate to the path mentioned above and edit this file

postgres@gwis:/etc/postgresql/16/main$ nano pg_hba.conf

scroll down until you see this section:

 # TYPE DATABASE USER ADDRESS METHOD

 # "local" is for Unix domain socket connections only

local all all trust

 # IPv4 local connections:

host all all 0.0.0.0/0 scram-sha-256

 # IPv6 local connections:

host all all ::1/128 scram-sha-256

 # Allow replication connections from localhost, by a user with the

 # replication privilege.

Make sure that the entries in bold have the values above.

postgresql.conf

In the postgresql.conf, there are two key configuration options which are important:

listen_addresses

This configuration defines which P address(es) to listen on. For standalone PostgreSQL servers the recommended setting is * (example below). This allows PostgreSQL to bind to any NICs presented to the server.

max_connections

This configuration defines how many connections PostgreSQL will allow. The default is 100 but the ba can either increase this if traffic is heavy or use connection pooling.

Where these lines are can vary but an expected single node Qlik Sense site would look like this:

listen_addresses = '*'

max_connections = 100

TABLESPACE (optional advanced topic)

A tablespace is a disk location where the physical data files of database objects (tables, indexes, sequences, etc.) are stored. Think of tablespaces as a mapping between a name and a directory on your server/computer. PostgreSQL comes out of the box with two tablespaces:

pg_default is the default tablespace used by PostgreSQL when you create a table.

pg_default maps to the /data directory in your PostgreSQL directory (defined by $PGDATA).

pg_global is used by PostgreSQL internals to store system-related objects.

sudo su – postgres

cd /var/lib/postgresql/16

cd main/base

ls

1 21613 4 5 pgsql_tmp

Note that 2163 directory is where the gwis database is stored

If you list the contents you will get all the database objects:

```
postgres@gwis:~/16/main/base/21613$ ls
112         23312_fsm   23446_fsm   24746      2617_vm    2837       3603_vm
113         23312_vm    23446_vm    24747      2618       2838       3604
1247        23315       23464       24748      2618_fsm   2838_fsm   3605
1247_fsm    23318       23467       24749      2618_vm    2838_vm    3606
1247_vm     23319       23468       24750      2619       2839       3607
1249        23330       23469       24751      2619_fsm   2840       3608
1249_fsm    23330_fsm   23470       24752      2619_vm    2840_fsm   3609
1249_vm     23333       23473       24753      2620       2840_vm    3712
1255        23333_fsm   23474       24754      2620_fsm   2841       3764
1255_fsm    23333_vm    23477       24755      2650       2995       3764_fsm
1255_vm     23334       23480       24756      2651       2996       3764_vm
1259        23335       23481       24757      2652       3079       3766
1259_fsm    23336       23484       24758      2653       3079_fsm   3767
1259_vm     23336_fsm   23487       24759      2654       3079_vm    3997
13442       23339       23488       24760      2655       3080       4143
13442_fsm   23339_fsm   23489       24761      2656       3081       4144
13442_vm    23339_vm    23492       24762      2657       3085       4145
13445       23340       23492_fsm   24763      2658       3118       4146
13446       23341       23492_vm    24764      2659       3119       4147
13447       23342       23515       24765      2660       3164       4148
13447_fsm   23342_fsm   23515_fsm   24766      2661       3256       4149
13447_vm    23345       23518       24767      2662       3257       4150
13450       23345_fsm   23521       24768      2663       3258       4151
13451       23345_vm    23524       24769      2664       3350       4152
13452       23346       23525       24770      2665       3351       4153
13452_fsm   23347       23528       2579       2666       3379       4154
13452_vm    23348       23529       2600       2667       3380       4155
13455       23348_fsm   23530       2600_fsm   2668       3381       4156
13456       23348_vm    23556       2600_vm    2669       3394       4157
13457       23351       23567       2601       2670       3394_fsm   4158
13457_fsm   23351_fsm   23568       2601_fsm   2673       3394_vm    4159
13457_vm    23352       23568_fsm   2601_vm    2674       3395       4160
13460       23353       23571       2602       2675       3429       4163
13461       23354       23572       2602_fsm   2678       3430       4164
1417        23357       23573       2602_vm    2679       3431       4165
1418        23358       24676       2603       2680       3433       4166
174         23359       24678       2603_fsm   2681       3439       4167
175         2336        24680       2603_vm    2682       3440       4168
2187        23365       24682       2604       2683       3455       4169
21932       2337        24684       2604_fsm   2684       3456       4170
21932_fsm   23371       24686       2605       2685       3456_fsm   4171
21932_vm    23374       24688       2605_fsm   2686       3456_vm    4172
21936       23375       24690       2605_vm    2687       3466       4173
21937       23375_fsm   24692       2606       2688       3467       4174
21938       23388       24694       2606_fsm   2689       3468       5002
2224        23389       24696       2606_vm    2690       3501       548
2228        23395       24698       2607       2691       3502       549
23270       23395_fsm   24700       2607_fsm   2692       3503       6102
23270_fsm   23395_vm    24702       2607_vm    2693       3534       6104
23273       23408       24704       2608       2696       3541       6106
23273_fsm   23408_fsm   24706       2608_fsm   2699       3541_fsm   6110
23274       23411       24708       2608_vm    2701       3541_vm    6111
23275       23411_fsm   24710       2609       2702       3542       6112
23276       23412       24712       2609_fsm   2703       3574       6113
23276_fsm   23413       24714       2609_vm    2704       3575       6116
2328        23414       24716       2610       2753       3576       6117
```

These are not accessible through OS commands. Each table and index is stored in a separate file. For ordinary relations, these files are named after the table or index's filenode number, which can be found in pg_class.relfilenode. In addition to the main file, each table and index has a free space map (see Section 73.3), which stores information about free space available in the relation. The free space map is stored in a file named with the filenode number plus the suffix _fsm. Tables also have a visibility map, stored in a fork with the suffix _vm.

```
postgres=#  SELECT dattablespace from pg_database where datname = 'gwis';
 dattablespace
---------------
          1663
```

This is the filenodenumber for the tablespace.

When a table or index exceeds 1 GB, it is divided into gigabyte-sized segments. The first segment's file name is the same as the filenode; subsequent segments are named filenode.1, filenode.2, etc

There are two main use cases that we can benefit from tablespaces.

First, tablespaces are a strategic tool to prevent storage from filling up. As your disks become full, you can assign tables or partitions to different disk locations to avoid database outages. It's important to note that only new data will be stored on the new disk, while the existing data continues residing on the old disk.

Secondly, and perhaps most importantly, tablespaces are a crucial tool in optimizing your PostgreSQL databases' performance and monetary cost. This is done by storing frequently accessed data on high-speed, expensive disks while moving less frequently accessed data to more cost-effective, slower disks. For instance, using time-series data, you could keep the most recent three days of data on a fast disk. Meanwhile, historical and reporting data older than three days could be moved to inexpensive, cold storage.

PART 4 – Geoserver

OVERVIEW

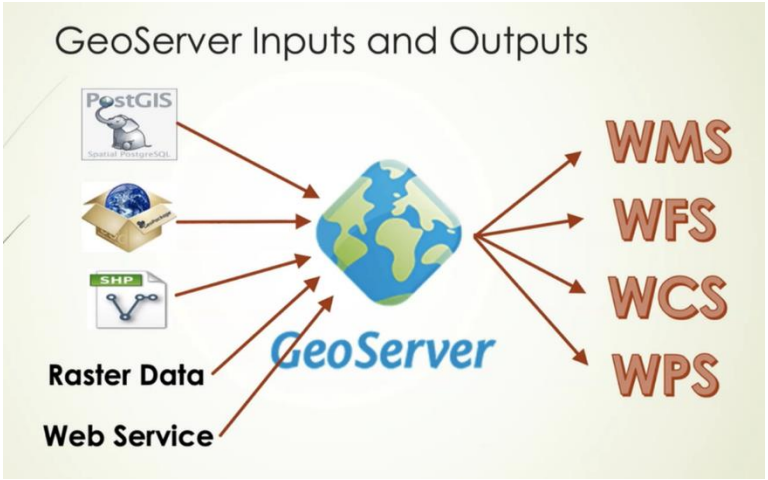
GeoServer is an open-source server for sharing geospatial data.

Designed for interoperability, it publishes data from any major spatial data source using open standards. GeoServer has evolved to become an easy method of connecting existing information to virtual globes such as Google Earth and NASA World Wind. It allows you to serve maps and data from a variety of formats to standard clients such as web browsers and desktop GIS programs. Data is published via standards-based interfaces, such as WMS, WFS, WCS, WPS, Tile Caching and more. GeoServer comes with a browser-based management interface and connects to multiple data sources at the back end.

One reason for proposing the inclusion of GeoServer, is it's Web Map Service (WMS) as a convenient way to handle raster datasets. While the PostgreSQL /PostGIS database can store raster data, the browser-based web clients (usually written in JavaScript) have difficulty displaying them. Geoserver delivers these datasets in a variety of image formats that can be easily rendered in browsers. It will serve as the main repository of digital elevation models and remote sensing products. Experienced users can connect to these data sources for image processing and analysis using a desktop GIS.

Users can create their own maps on their own GIS and computers, then publish them on the map server to share online.

The other aspect of Geoserver is enhanced interoperability. Connections can be made to ICPAC sites such as The East Africa Drought Watch and real-time runoff forecasting, and the ICPAC Geoportal, and IGAD Geonode.



GeoServer has the ability to proxy a remote Web Map Service (image data) and Web Feature Service (vector data) This process is sometimes known as Cascading WMS/WFS. Loading a remote WMS is useful for many reasons. If you don't manage or have access to the remote WMS, you can now manage its output as if it were local.

WMS cascading allows to expose layers coming from other WMS servers as if they were local layers. Clients connecting to your SDI need to care about less points of origin, which might be important for high security networks

It is now possible to ask for maps in formats not supported by the original server, or to reproject the maps in projections not supported by the original server. In this way, the GWIS can connect to other databases, and the administrators of other databases can connect to the GWIS.

Geoserver comes with its own layer of security. GeoServer supports access control at the service level, allowing for the locking down of service operations to only authenticated users who have been granted a particular role. OWS services support setting security access constraints globally for a particular service, or to a specific operation within that service. There are also security settings related to data management and Layer security. Data access is granted to roles, and roles are granted to users and groups. Therefore, a system exists outside of the main application which can control what users can and cannot do.

Prerequisites

When migrating, the server administrator must install Java Development Kit, or at least Java Runtime Environment, and Apache Tomcat. Instructions are not given here, as they vary depending on the operating system and software version. This is the responsibility of the IT department.

OGC standards

The chief product, Web Map Service, WMS, returns data as a geo referenced image or more commonly, a set of *tiled* images.

These may be background maps similar to what you would get from Google Maps or Open Street Maps, but they would be generated with your own custom data and most likely cover a relatively small area rather than the entire world.

The source data can be raster format such GeoTIFF, ImageMosaic, ArcGrid, Mr. Sid or other formats, especially ideal for large rasters.

There is also a special service called Web Coverage Service, WCS, which also handles rasters.

Geo Server receives a request for imagery and then reads the source data and returns the image as map tiles the size of which can be configured.

Maptiles are more efficient than a single large image because you only get the part of the image that

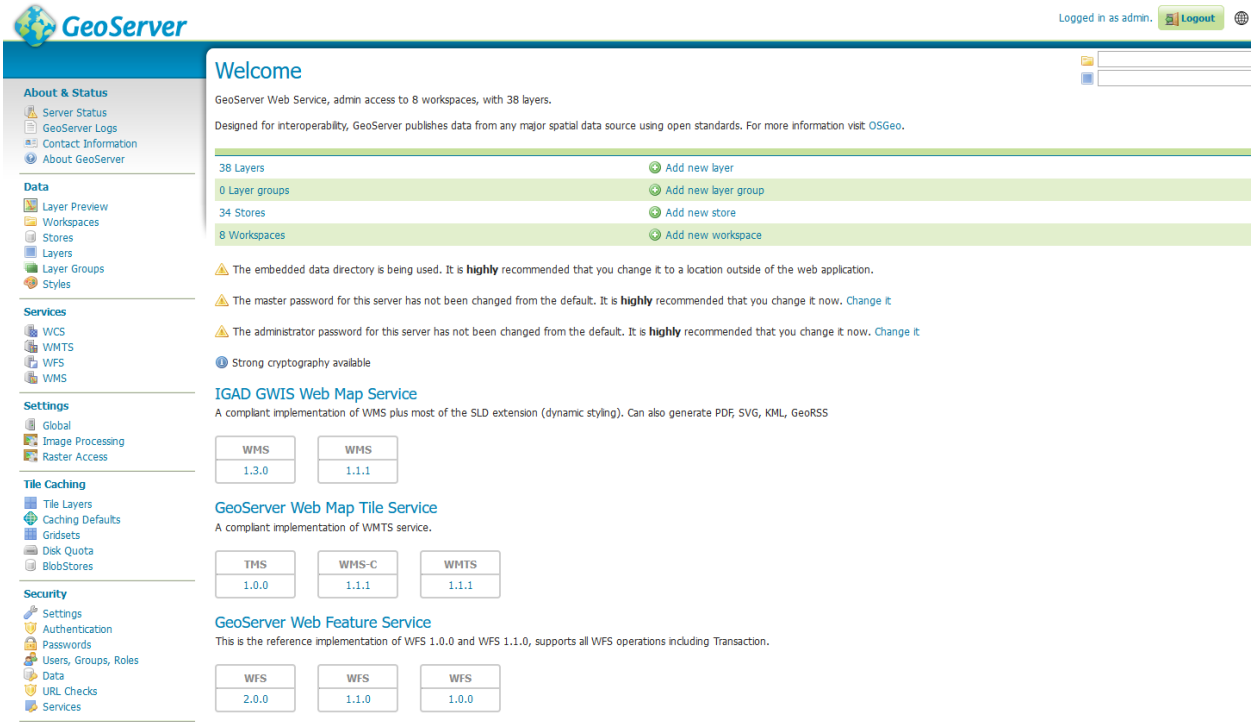
at appropriate resolution e.g. a one-meter resolution aerial photo. The tiles that are returned can be in JPEG, GIF, or PNG format. WMS can also convert vector data to images for display purposes. But they cannot be edited or used in vector-based analysis. If you want to be able to edit your data or analyze it, you need to use a WFS - Web Feature Service.

Access to Administration

To get to the Geoserver interface type http://197.254.1.11:8080/geoserver/

Log in with default username 'admin' with password of 'geoserver.' Note that the staff administering Geoserver should change passwords for both admin and master. This will be done during the training session. Ordinary users do not need to login, but they will only see the Layer Preview page.

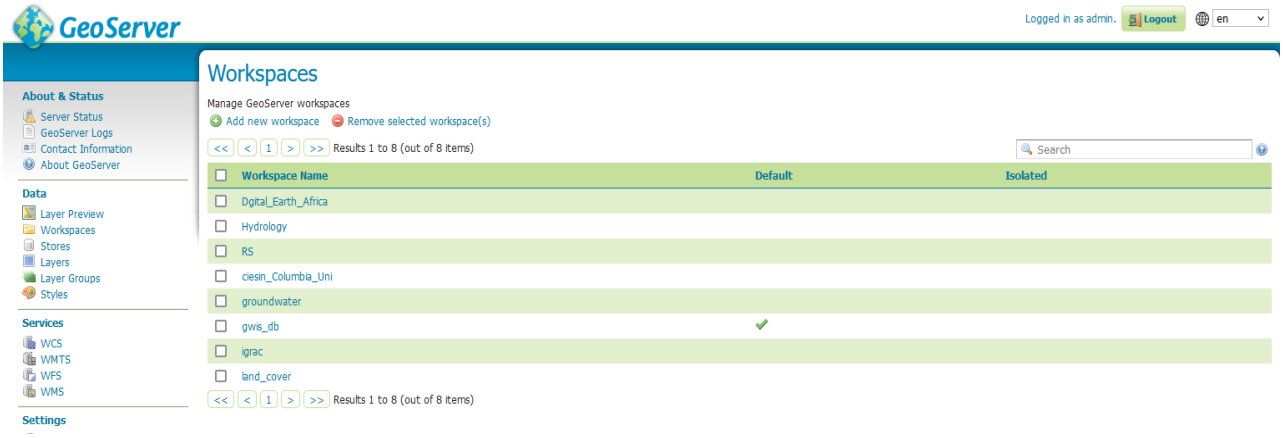
Once you have logged in as an administrator, many more items than Layer Preview appear in the side panel:



Note the different sections. If you click on the last item in the first section, "About Geoserver," and scroll to the bottom, you will see "Useful Links" and there is a direct link to the online documentation which is highly recommended reading, as it covers much more than this introductory material. Under "Contact Information", the appropriate details and contact person will be filled min during training.

Workspaces

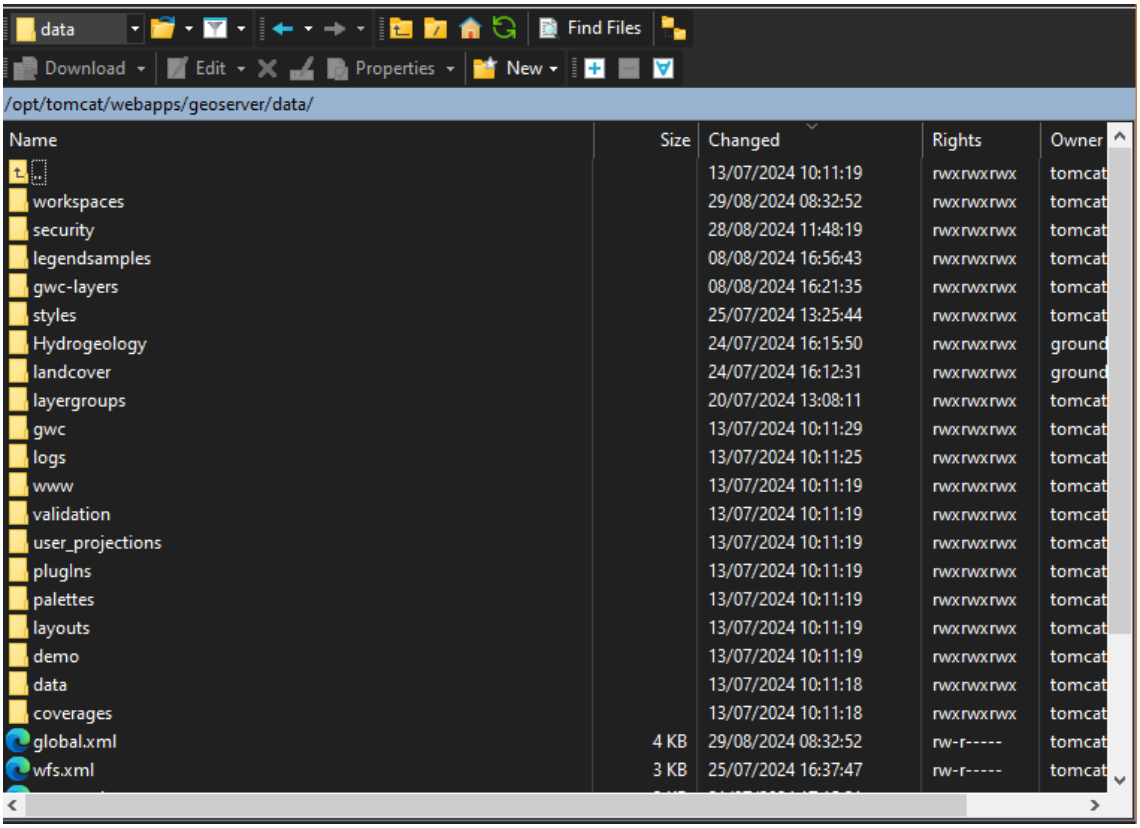
Workspaces help to organize datasets. For the GWIS, the consultant has created 8 workspaces as shown below, based on themes:



Three of these workspaces actually contain data from external map services: Digital Earth Africa, ciesin at Columbia University, and IGRAC.

Data Directory

The path to the data directory is **/opt/tomcat/webapps/geoserver/data** and is shown below in WinSCP

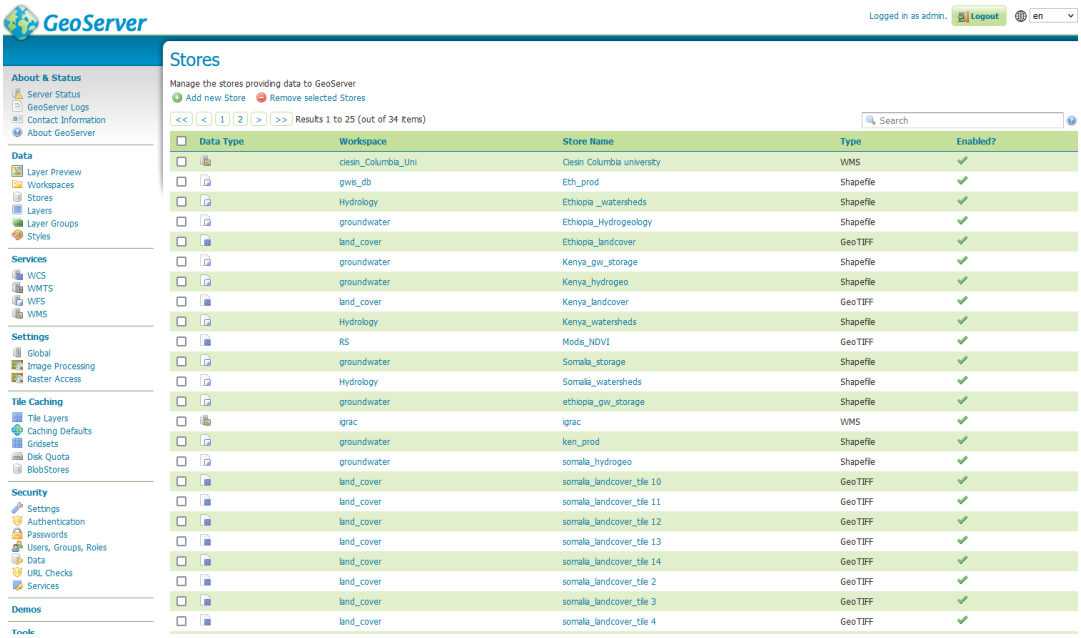


This is where Geoserver gets its maps. New data can be uploaded anywhere in this folder. When you add a new data source for display in Geoserver, you will be asked to browse for the file in this directory.

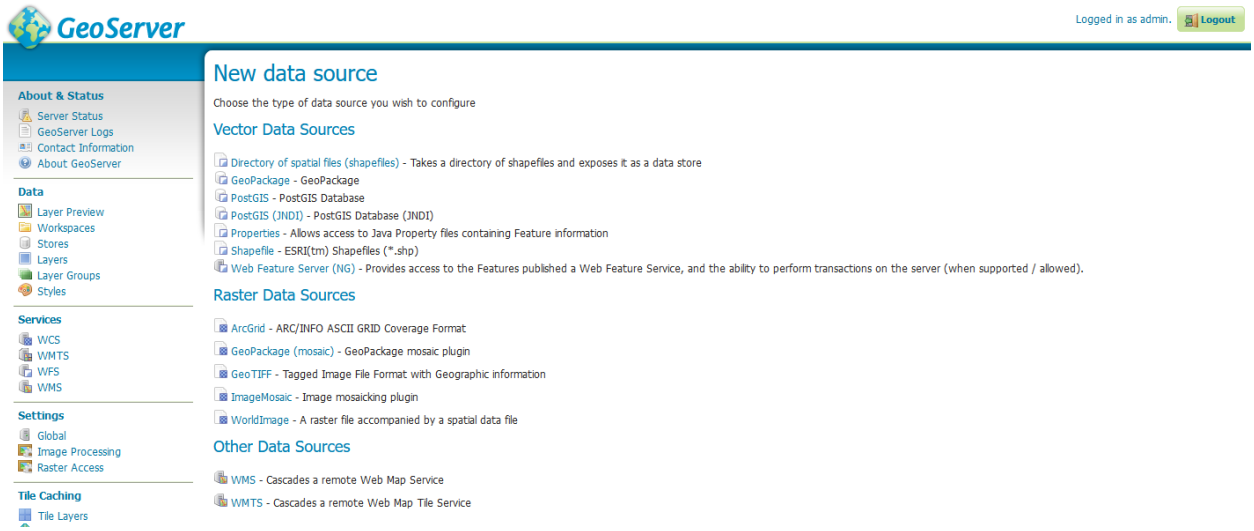
THE FIRST THING YOU MUST DO TO PUBLISH MAPS ON GEOSERVER IS TO UPLOAD THE DATA TO THE SERVER FIRST. It is also recommended to upload **Styles** you plan to use before publishing layers – see section on Styles.

Stores

Stores are data sources. There are 34 stores so far in the IGAD Geoserver.



If you click on "Add new store" you can view the various datatypes and connections that are offered.

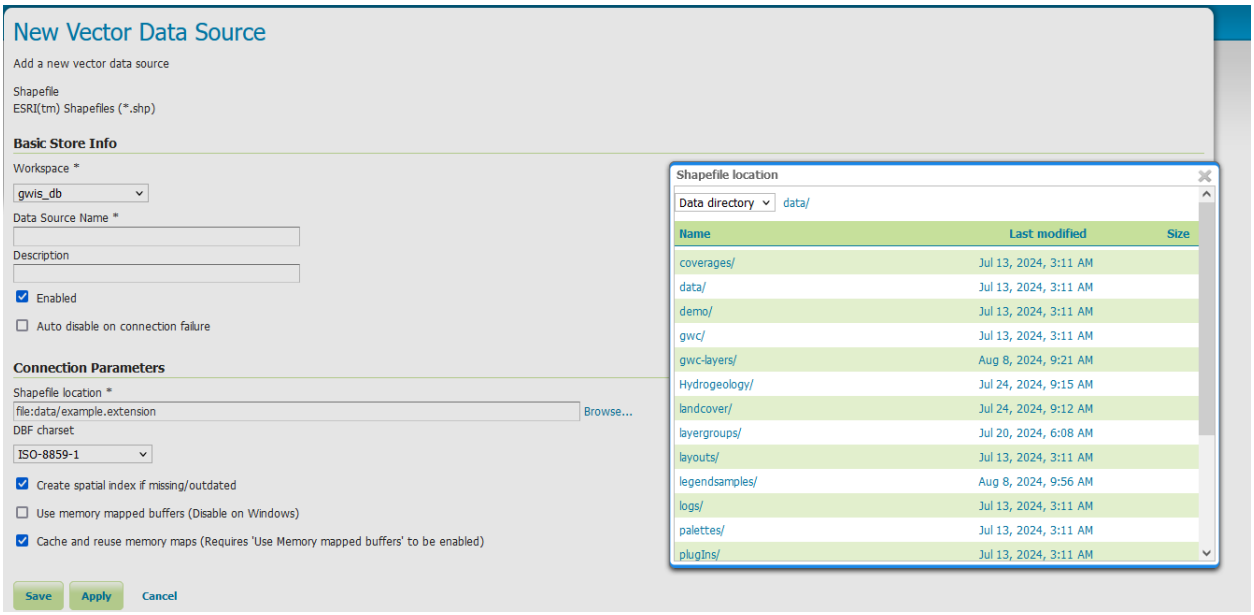


Note the last group "Other Data Sources." If you click on this and provide a special URL, called the getcapabilites URL, the system can get data from external servers. Below is an example of the Connection Info to connect to IGRAC:

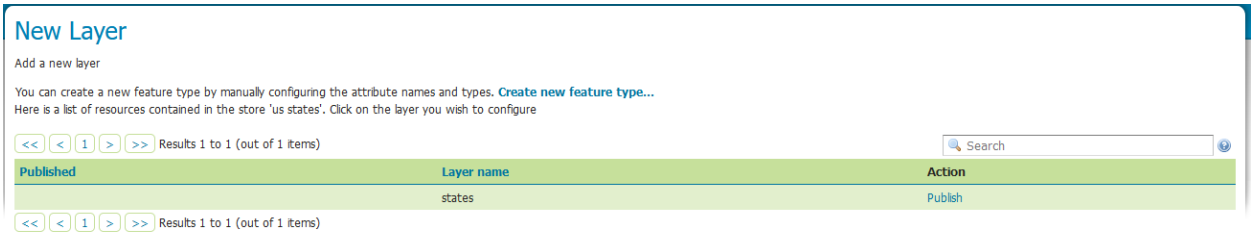
https://ggis.un-igrac.org/geoserver/ows?service=WCS&acceptversions=2.0.1&request=GetCapabilities

More about this will be mentioned under the section 'WORKFLOW."

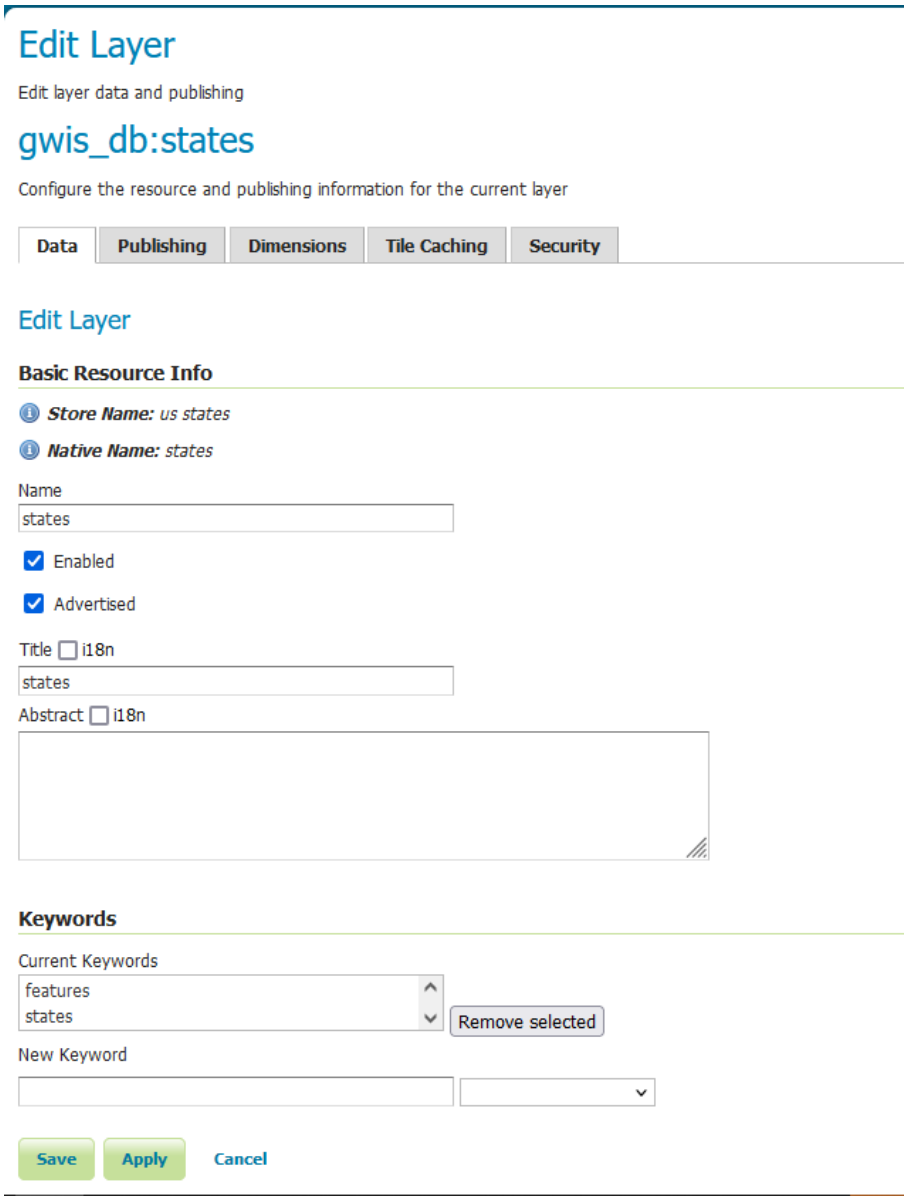
When you click on the type of data source you want to add, you will get this page:



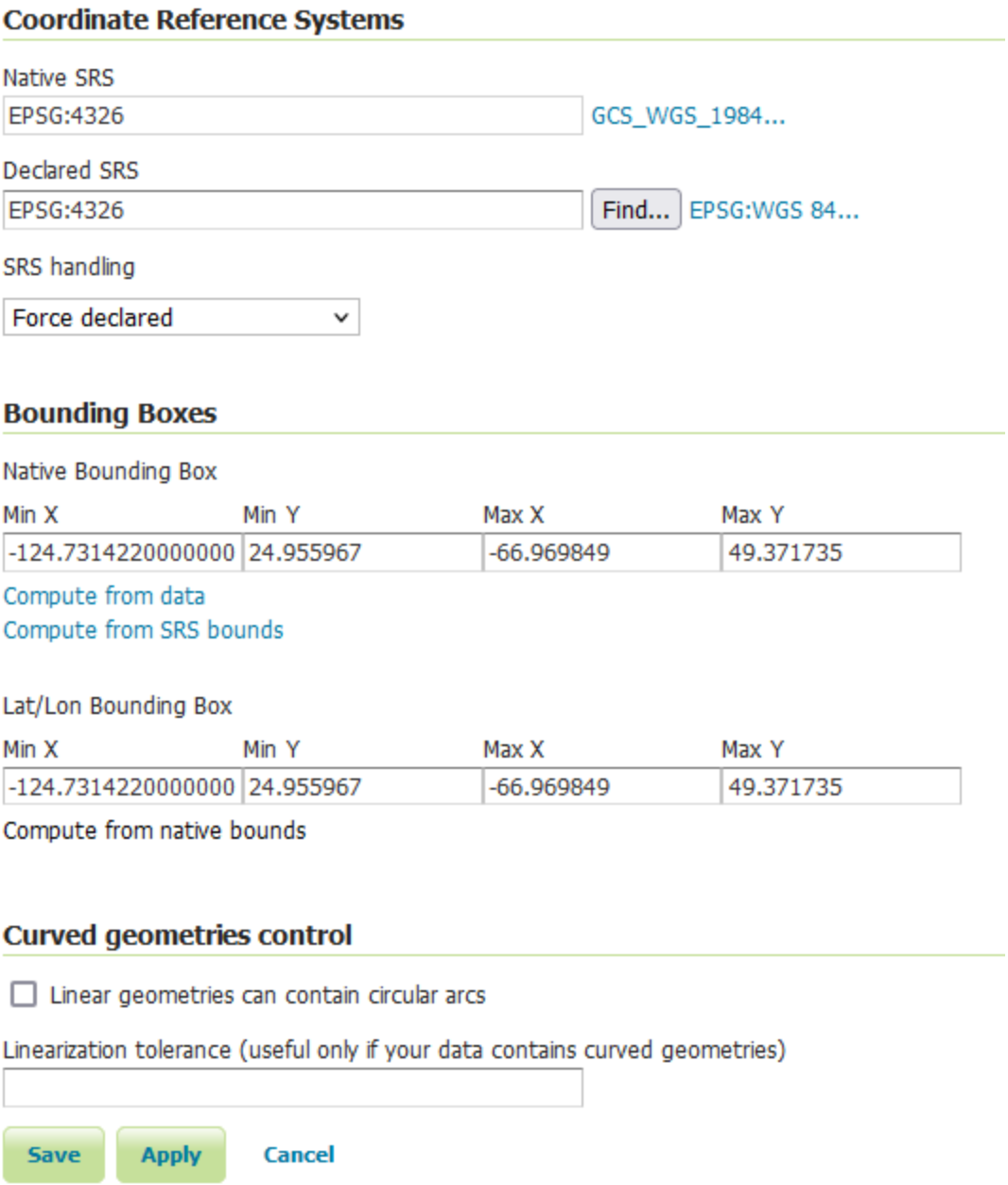
You must choose a workspace from the dropdown list. Click on "Browse" next to Connection parameters to bring up browse screen, which show *the data already uploaded to the data directory* on the server. After choosing the file, enter the name and description. Click save and the screen changes:



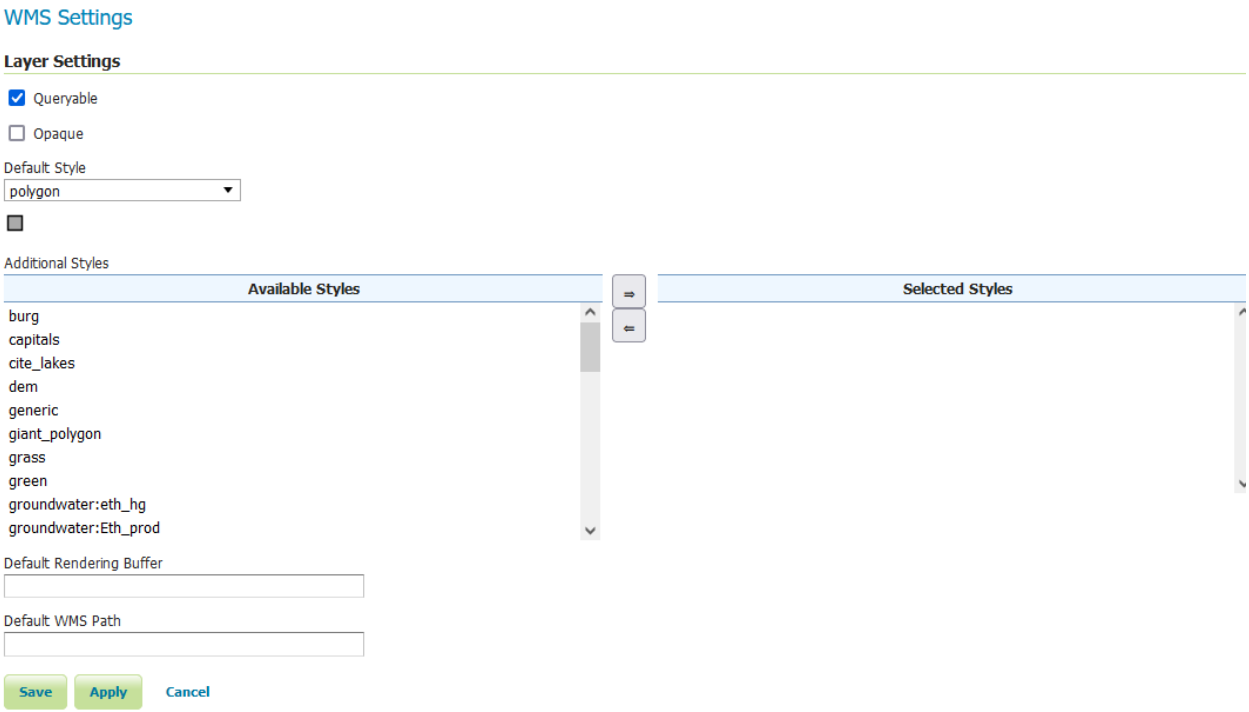
Under the heading **Action**, click **Publish** This brings you to Edit Layer page:



Fill in name, title, abstract. Towards the bottom important, mandatory details concerning CRs and bounding boxes must be filled in – click on Compute From Data and for the second, click on Compute from Bounds



Click on the publishing table and choose a style. The available styles are styles already in the system. More on creating your own styles later.



Click save. The Layers page is then presented.

LAYERS

You can publish a data source so that it is available as a layer. At the moment there are 38 layers in the IGAD Geoserver.



Layer groups – individual layers can be combined into groups to be displayed together as one layer.

Styles

There are many default styles that come with Geoserver installation but many of them are not very pleasing to look at. For vectors which are retrieved through WFS, this is not important since these layers can be brought into a GIS and styled according to the user's preferences. Various styles, such as landcover, were created by the consultant.



Styles

Manage the Styles published by GeoServer

Add a new style Removed selected style(s)

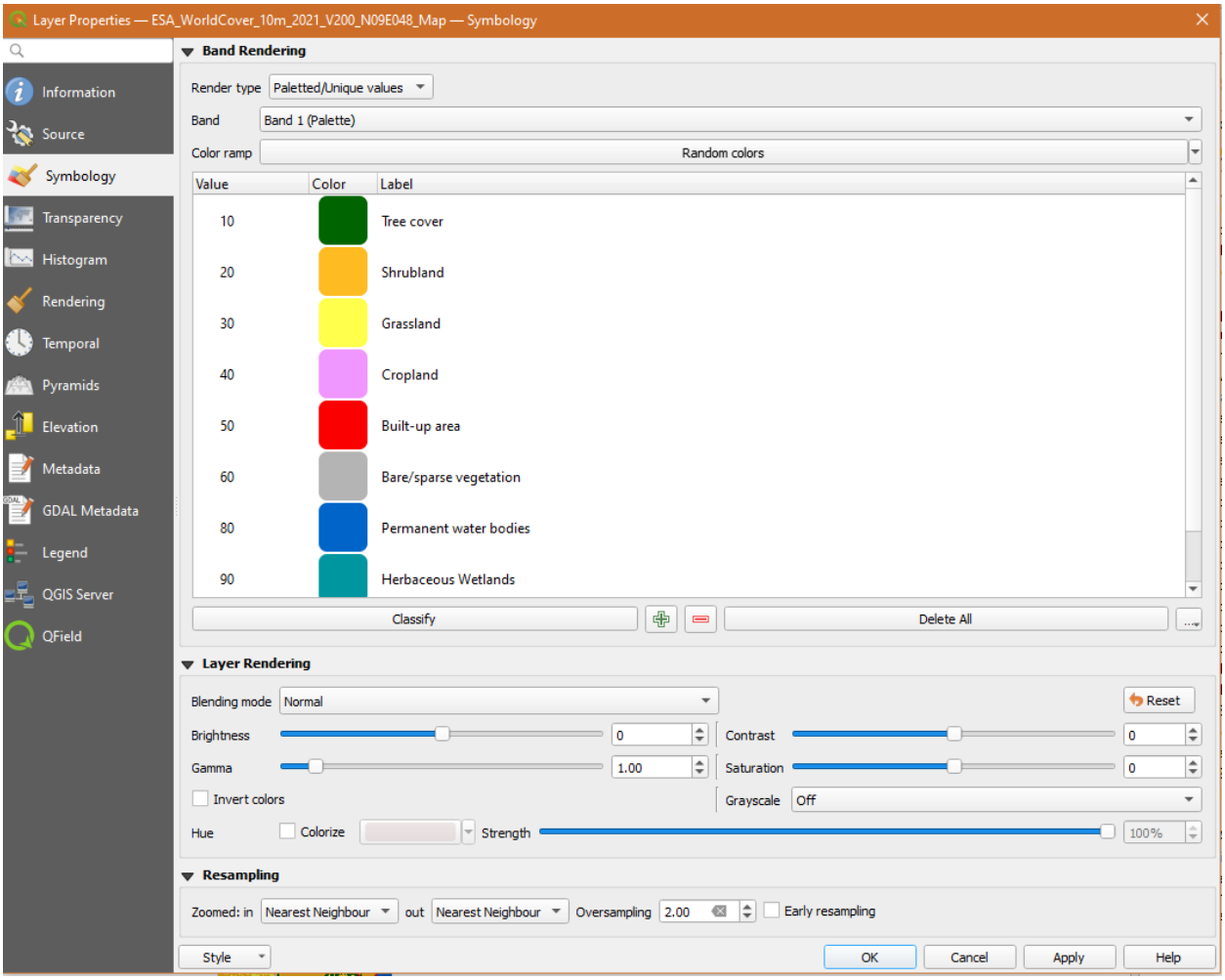
<< < 1 2 > >> Results 1 to 25 (out of 34 items) Search

Style Name	Format	Workspace
Eth_prod	SLD 1.1	groundwater
Ken_prod	SLD 1.1	groundwater
Som_prod	SLD 1.1	groundwater
burg	SLD 1.0	
capitals	SLD 1.0	
cite_lakes	SLD 1.0	
dem	SLD 1.0	
eth_hg	SLD 1.1	groundwater
eth_store	SLD 1.1	groundwater
generic	SLD 1.0	
giant_polygon	SLD 1.0	
grass	SLD 1.0	
green	SLD 1.0	
ken_store	SLD 1.1	groundwater
kenya_hg	SLD 1.1	groundwater
landcover	SLD 1.0	land_cover
line	SLD 1.0	
ndvi-green	SLD 1.0	RS
poi	SLD 1.0	
point	SLD 1.0	
poly_landmarks	SLD 1.0	
polygon	SLD 1.0	
pophatch	SLD 1.0	
population	SLD 1.0	

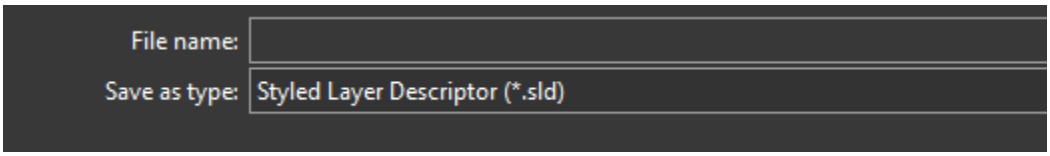
To create your own styles you can write an XML file, but an easier way is to use QGIS.

Bring in the data into QGIS

Double click on the layer to bring up the layer Properties screen. Click on Symbology in the left panel.



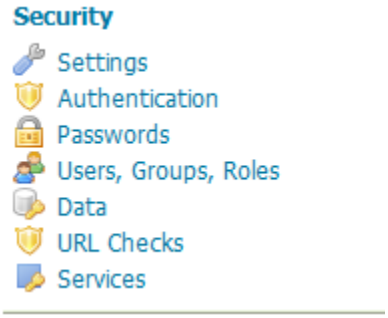
Notice the Style button at the bottom left. After creating the style you want, click on this button. In the context menu that appears, choose Save Style – a save dialogue box appears choose Styled Layer Descriptor.



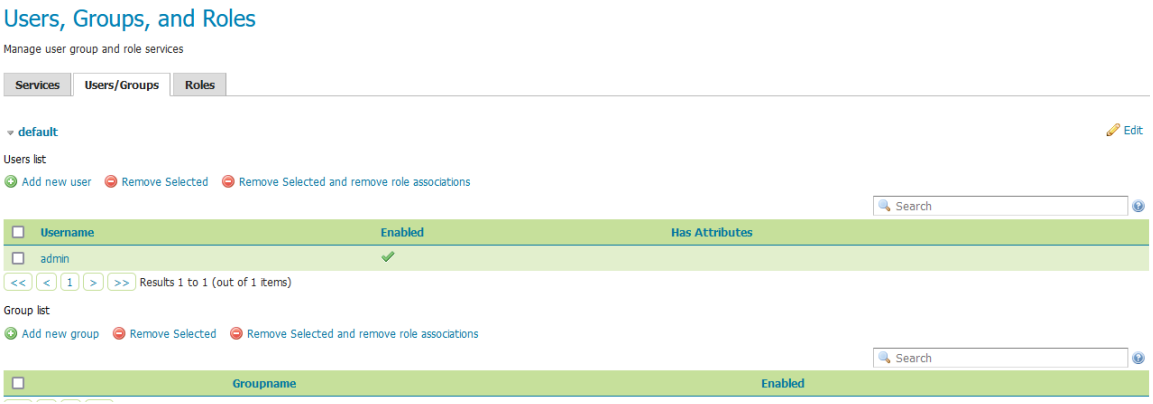
Find that file and upload to server in the styles subdirectory

/opt/tomcat/webapps/geoserver/data/styles

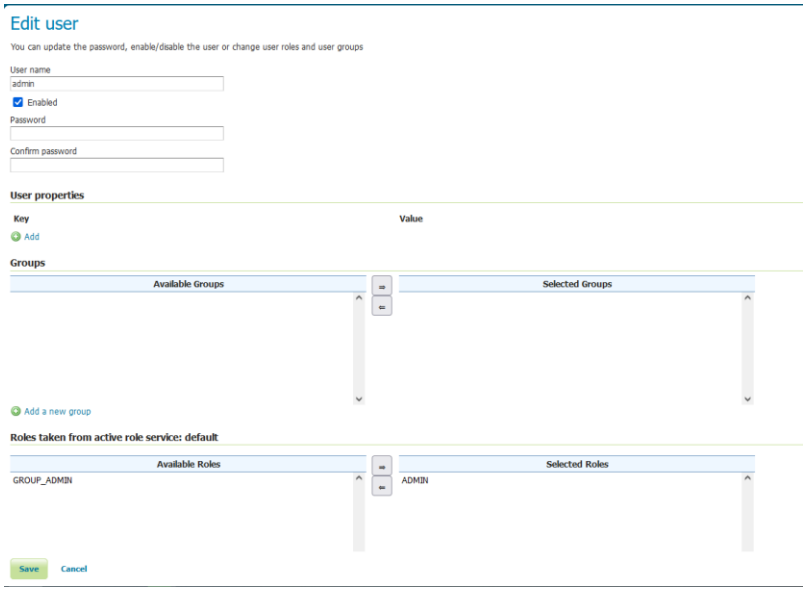
SECURITY



Most of these options are advanced and administrators may want to find out more through the documentation. What is important for controlling access is the creation of users, groups, and roles. Click on that, then click on the users/groups tab:



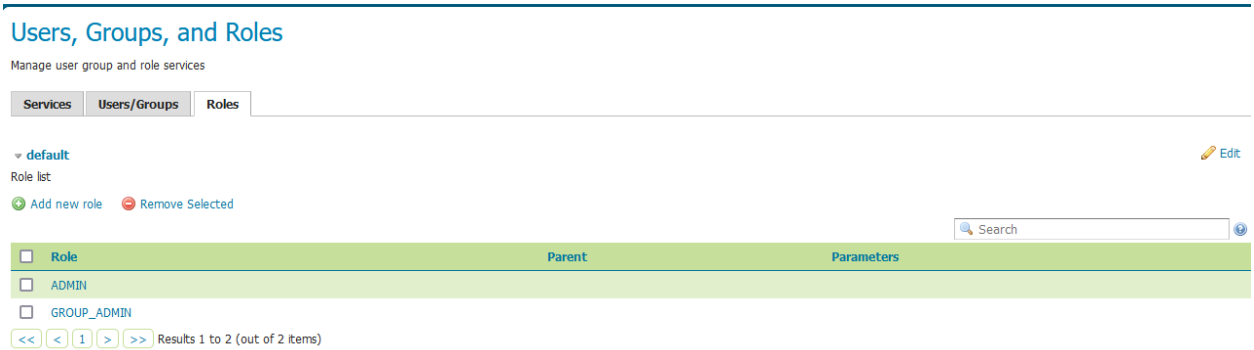
Note that we can add users on this page -see the green button on the right. We see there is only one user listed at the moment – admin. Click on that to see its properties



Look on the lower right we see that the admin user has the admin role as selected role

Go back to the previous page and click on Roles:

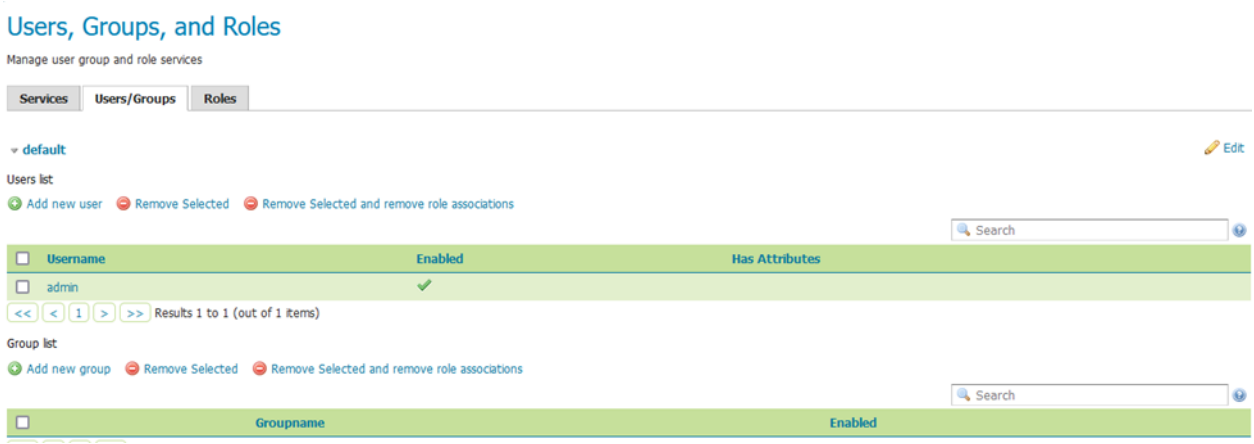
We see there 2 roles, ADMIN and GROUP_ADMIN



Click on add new role to add a new role

Give it a name and click save

Now we see this new user is listed.



Click on Add new group

Add roles to the new group

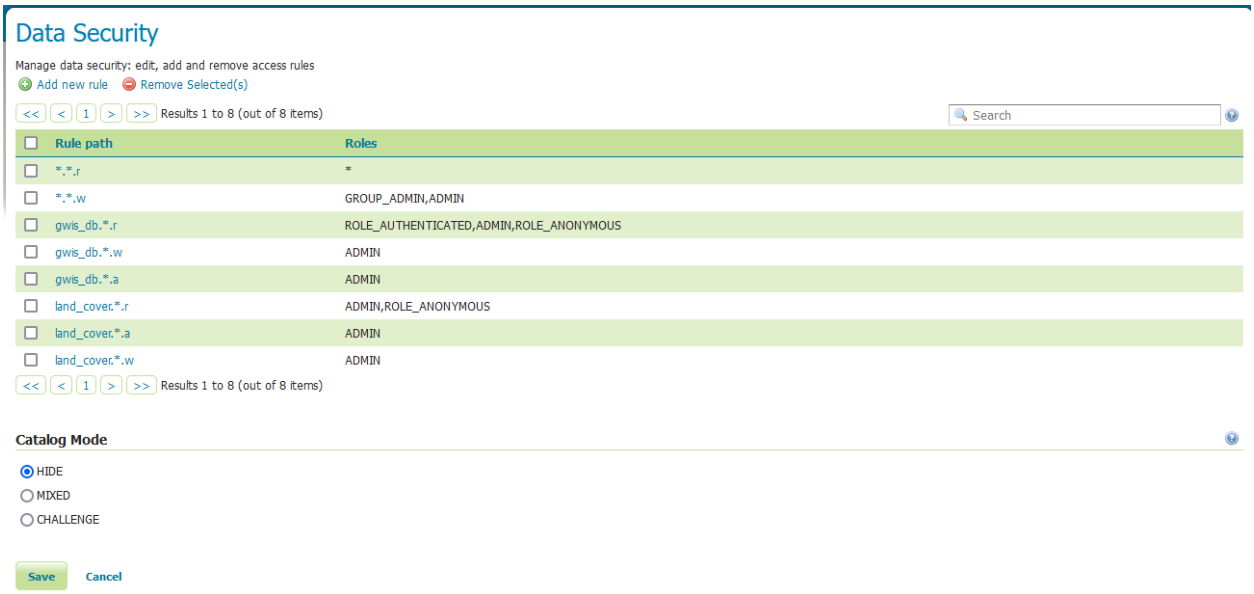
Then you can assign new users to groups The order of creation here is

Groups-> Roles -> privileges

Assigning a user to a group, they get the privileges of that group.

Or you can assign roles to users directly (easier!)

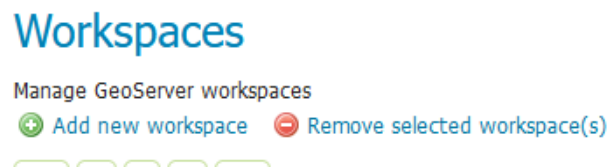
To define these privileges go to back a page to the Security section and click Data in the Security section



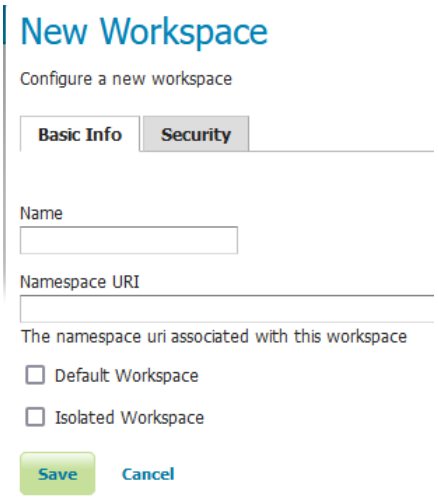
First asterisk means any workspace, second asterisk means any layer. The r means read, w means write.

SUMMARY OF WORKFLOW

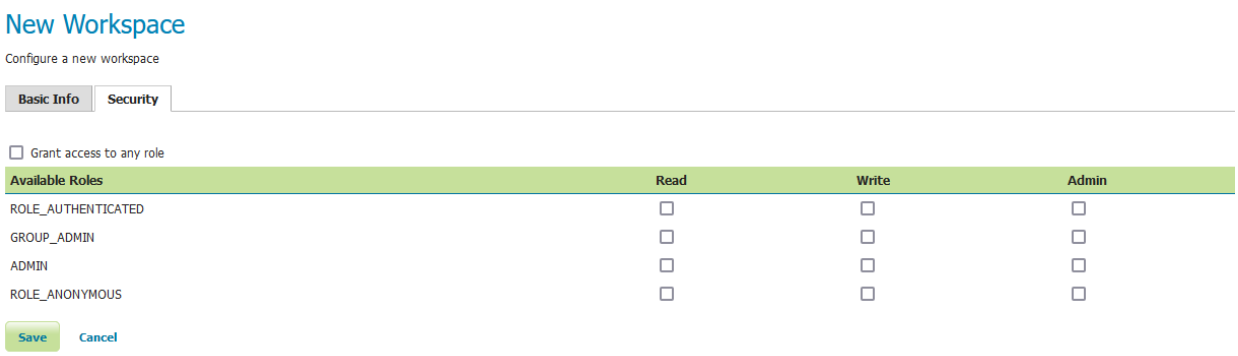
The first administrative task is to choose a workspace for a new layer. If the administrator feels that none of the existing workspaces are appropriate, a new one can be created. Click on "add new workspace."



Fill in a name you like – but NO spaces. The uri can also be anything. Leave default workspace and isolated workspace blank.



Click on security tab



create ROLES first then ADD USERS with names and passwords

assign roles to users - go into edit workspace and assign privileges: read, write admin to the roles

data must be uploaded ito the Geoserver data directory - FIRST THING

Stores - add data stores - new data store choose type format new source - choose workspace

if postgis must supply connection parameters

localhost, port, database name, user, password, public schema

for otherdata sources fill in data source name and description and browse for file

save will take you add new layer here's where you publish layer (otherwise layers-add layer)

edit layer publishing page

to publish exisitng layers go to layers-add layer)

data tab - check crs bounding boxes - compute from data (native bounding box)and compute from native bounds (lat/long bounding box)

publishing tab

go down to WMS setting and choose default style

then save

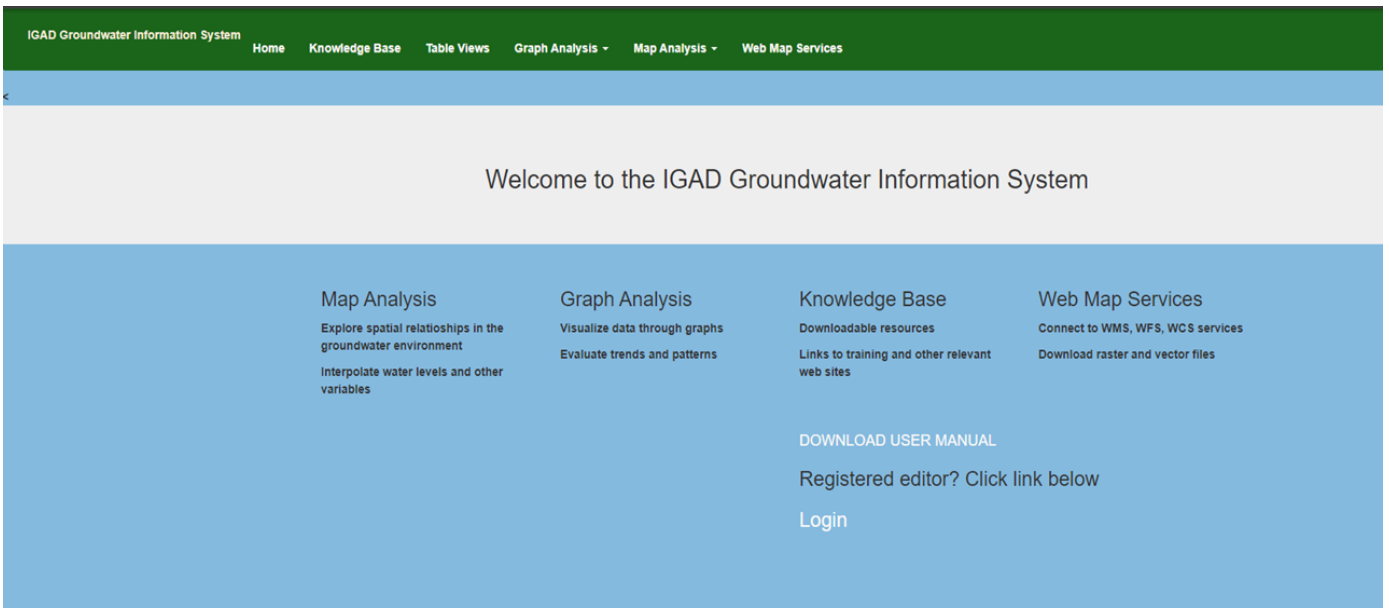
Group Layers

Go to layers add layer and browse. You can change order and styles. Make sure to generate bounds.

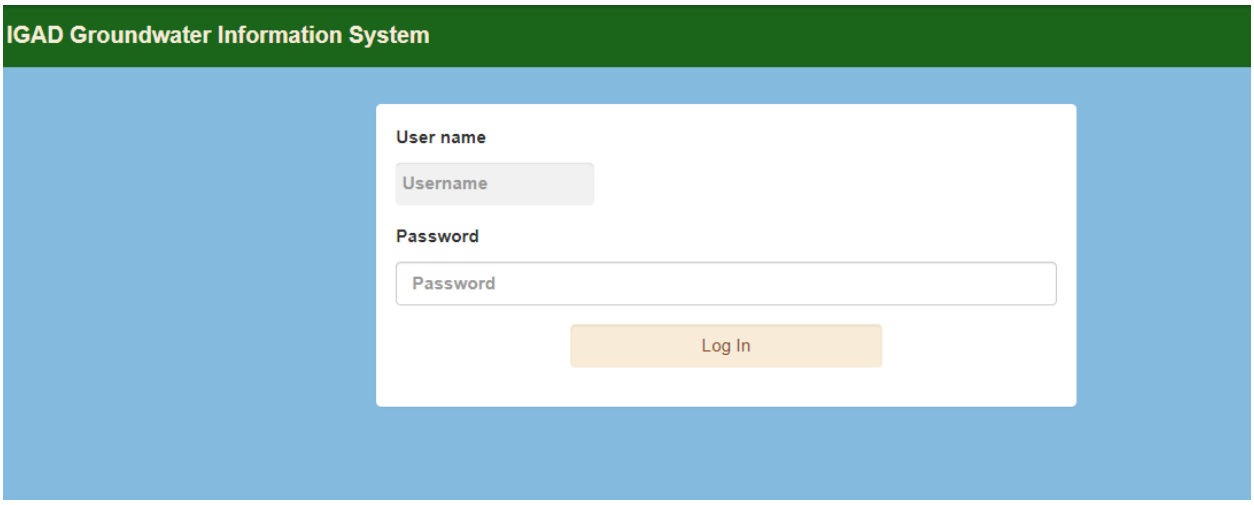
PART 5 -ADDING DATA TO THE SYSTEM

Browser Forms for record-by-record data entry

The GWIS has a browser based mechanism for entering information into the database. In order to gain access to this module, a user needs to log in through a link at the bottom of the entrance page.



Click on this link brings you to the login page

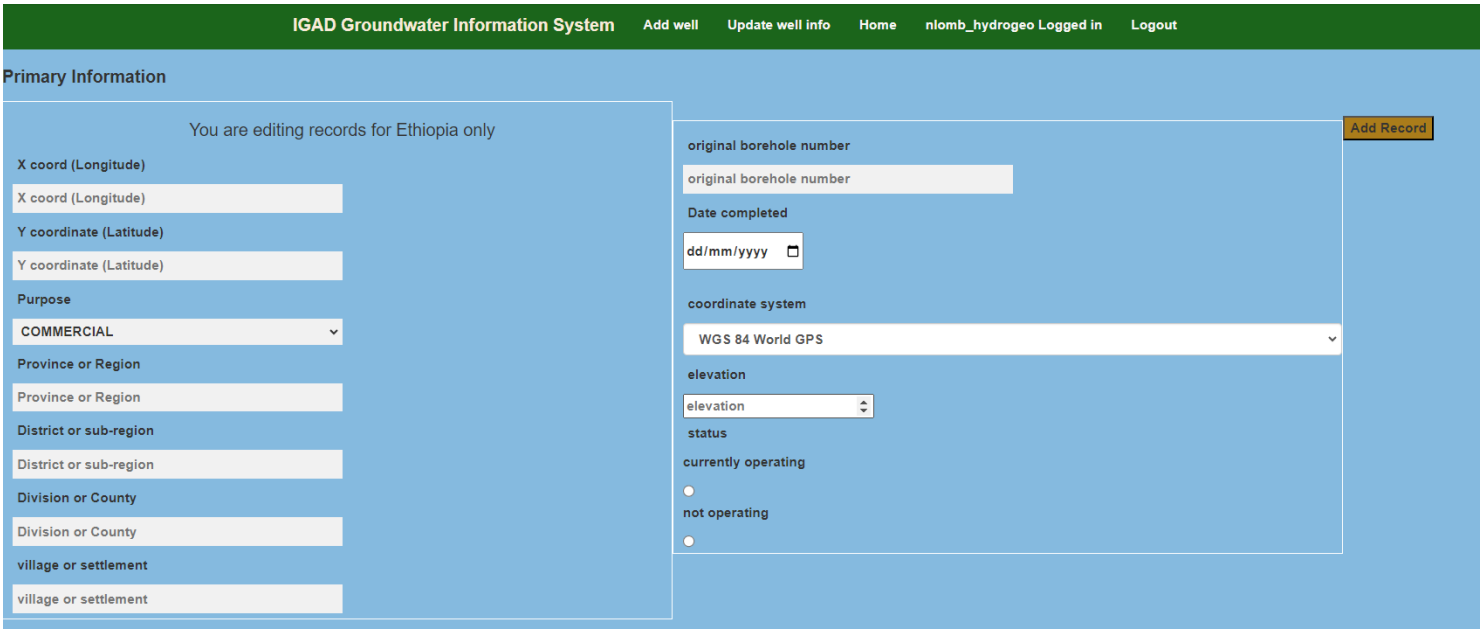


In order to get past this page, the user will need to be assigned a username and password by IGAD. The IGAD database administrator will have entered this information in a users table. The code behind the login page checks the information supplied by the userto see if it exits, and whether the user has an active status (active = 1)

Users table:

	fullname character varying (100)	username character varying (25)	password character varying (255)	joined date	active smallint	country character varying (100)	joined date	last_login date
1	nicholas Lom	nlomb_hydrogeo	$2y$10$fCrvDHVg0psXg8PibO9WtuoAongaflPrzzV1Gq8HSwN73bkr.lBGe	2024-07-01	1	Ethiopia	2024-07-01	2024-07-01
2	nikol coo	asdergh	$2y$10$3BQsdqpykBsRKFlX6jf1dOJnmwih1lFaaWR0mlXHAVpc9MSUTjn...	2024-07-02	[null]	Ethiopia	2024-07-02	2024-07-02

If the user exist in the table with active =1 the code direct to the first page of the form which pertains to the primary table – **well_parent**

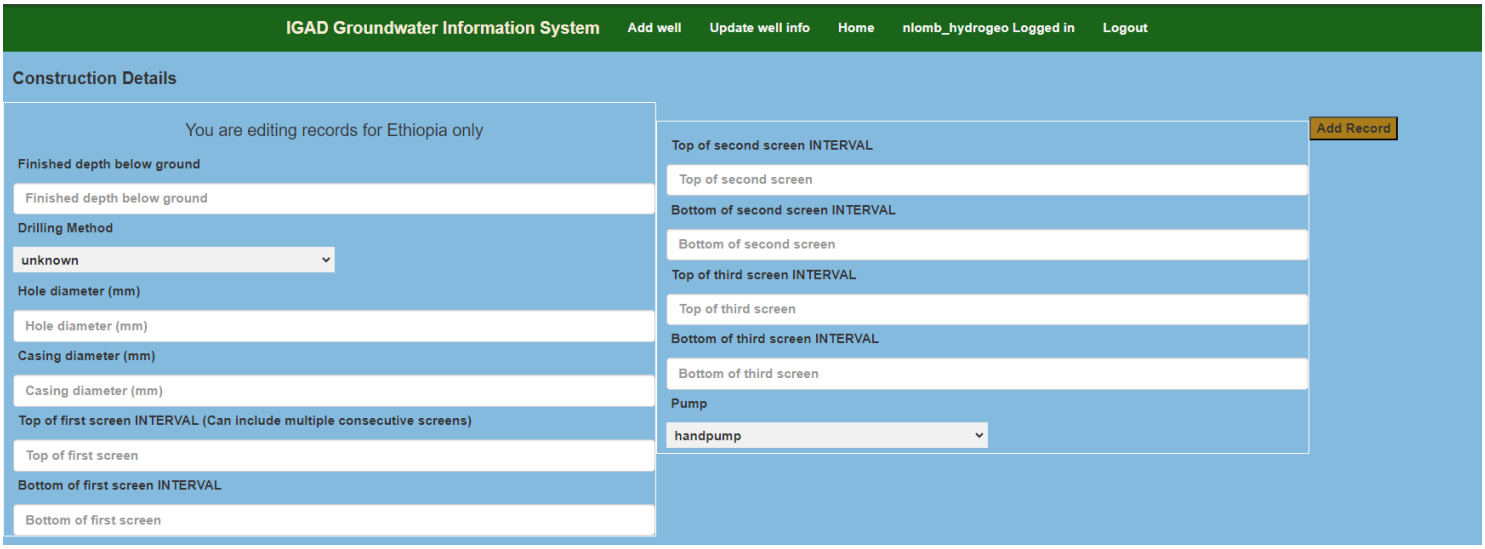


Primary information consists of location (including coordinates), purpose, original borehole number (required), date completed (required), coordinate system of the coordinates, elevation, and status.

Special features to note:

- The navigation bar at the top, next to last item on the rights, is the name of the user logged in.
- The message at the beginning of the form informs the user for which country records can be entered. This information is in the users table.
- The 'Add Record' button on the extreme right when the user has completed filling in the form
- Purpose and coordinate system are dropdown lists from which a choice is made. They cannot be blank and default values are shown
- The date completed is a date picker to choose the date. This also cannot be blank. The user will get a message to choose a date or the 'Add Record' button will not function

When the 'Add Record' button is clicked the user is presented with the next form that deals with construction details:



IMPORTANT NOTE – USER MUST CLICK ON THE ADD RECORD BUTTON EVEN IF THERE'S NO INFORMATION TO BE ADDED. THIS ENSURES THAT LEAST THE WELL NUMBER IS INSERTED IN THE CONSTUCTION TABLE AND A RECORD IS INSERTED THAT AN BE UPDATED LATER. **DO NOT EXIT THE EDITING PROCESS UNTIL ALL 4 FORMS HAVE BEEN ADDED**.

This page has data validation features that ensure:

- Hole diameter is greater than casing diameter
- Bottoms of screens are below tops of screens, but above the total depth.

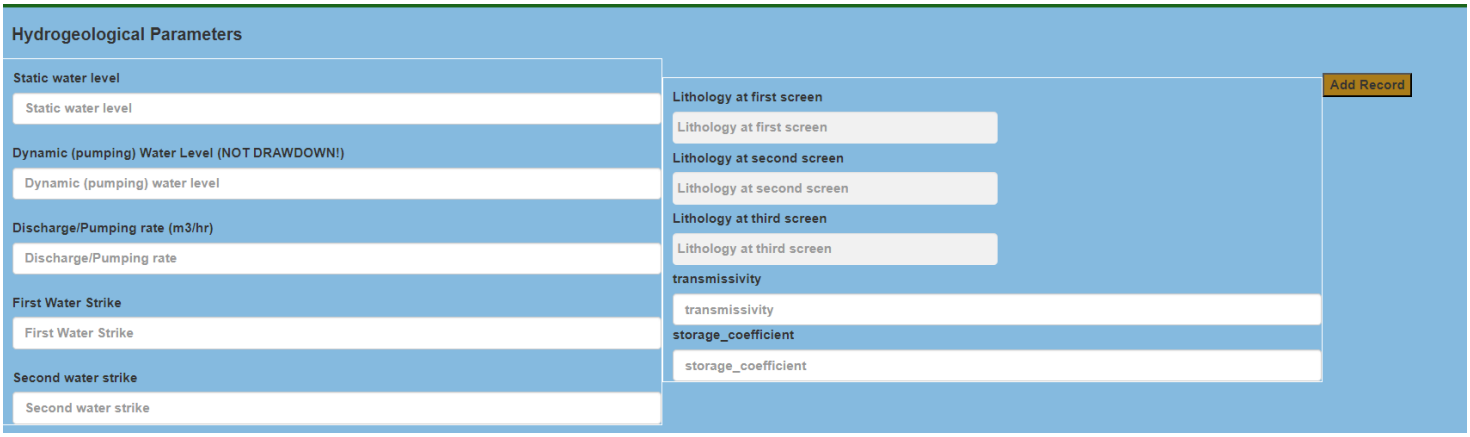
Note that screen positions are intervals, i.e. individual screens are not conisered.

Example first screens are at 24-27, 27-30, 33-36. This is considered as one interval, so for first screen the top is 23 and the bottom is 36. If screens are separated by a large distance, say 6 -10 meters, that could be considered a separate interval. This practice eliminates unnecessary redundant detail that has little bearing on aquifer position.

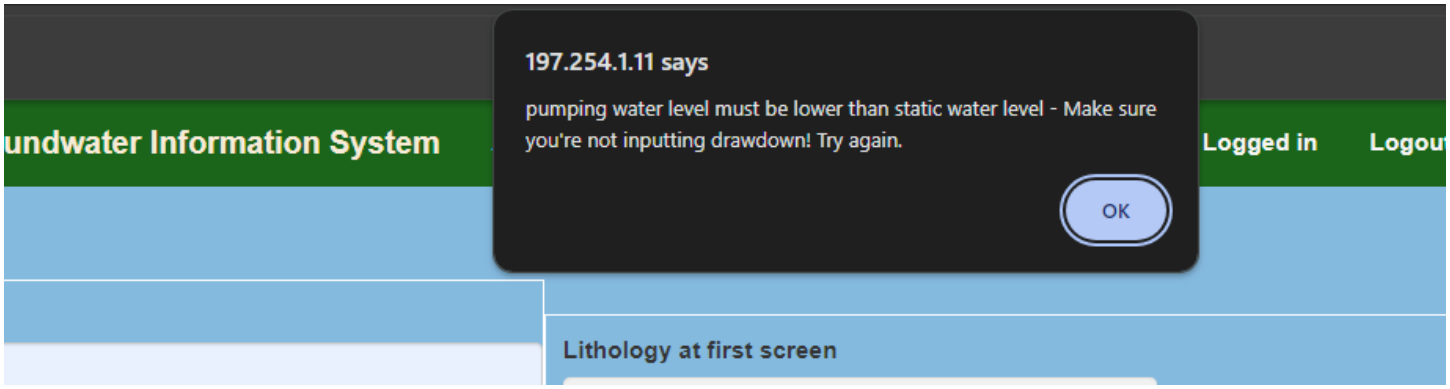
Drilling method and pump are dropdown lists.

Click the 'Add Record' button for the next form. Here, enter characterisitcss pertaining to water levels and strikes, lithologies adjacent to screens, transmissivity and storage (from pumping tests)

Specific capacity (drawdown divided by discharge) is automatically calculated by the database.

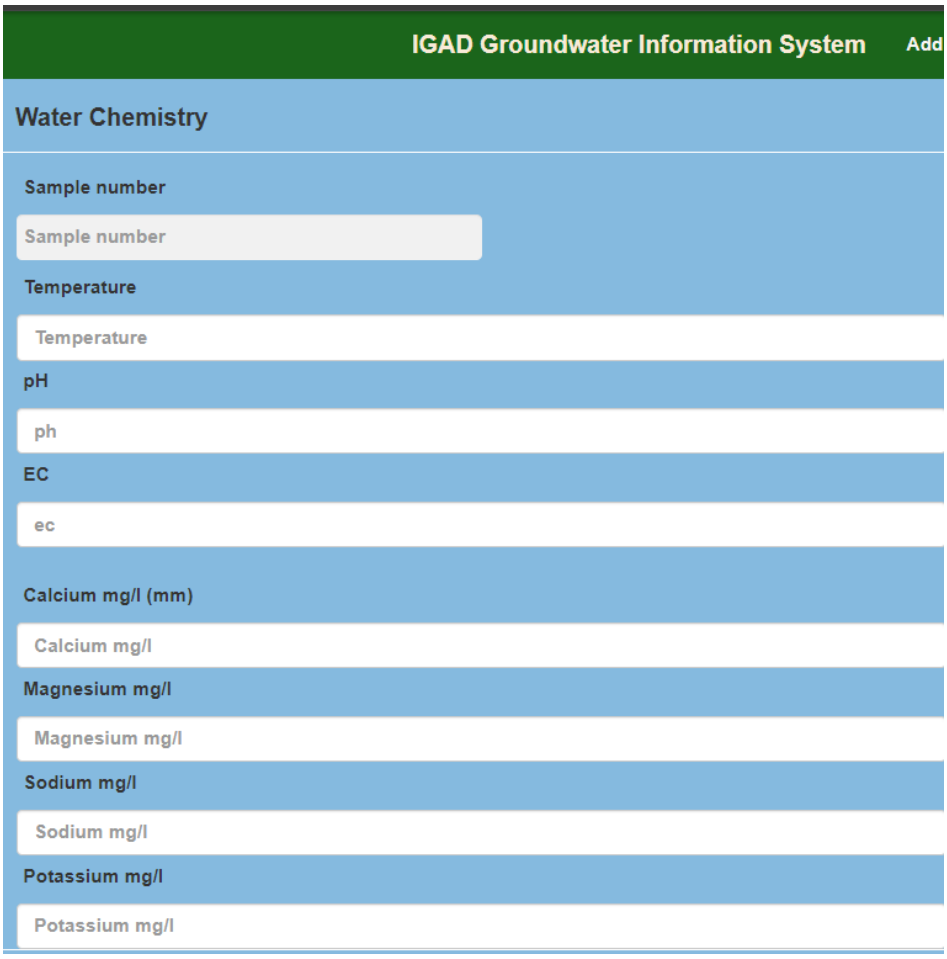


Here also exists data validation rules. Example is shown is if someone enters a pumping water level above the static water level. An alert pops up in such a case.



Some users get confused and input drawdown instead of DWL

Click the 'Add Record' button for the next form - Water Chemistry/Quality.


IGAD Groundwater Information System
Add
Water Chemistry
Sample number
Sample number
Temperature
Temperature
pH
ph
EC
ec
Calcium mg/l (mm)
Calcium mg/l
Magnesium mg/l
Magnesium mg/l
Sodium mg/l
Sodium mg/l
Potassium mg/l
Potassium mg/l

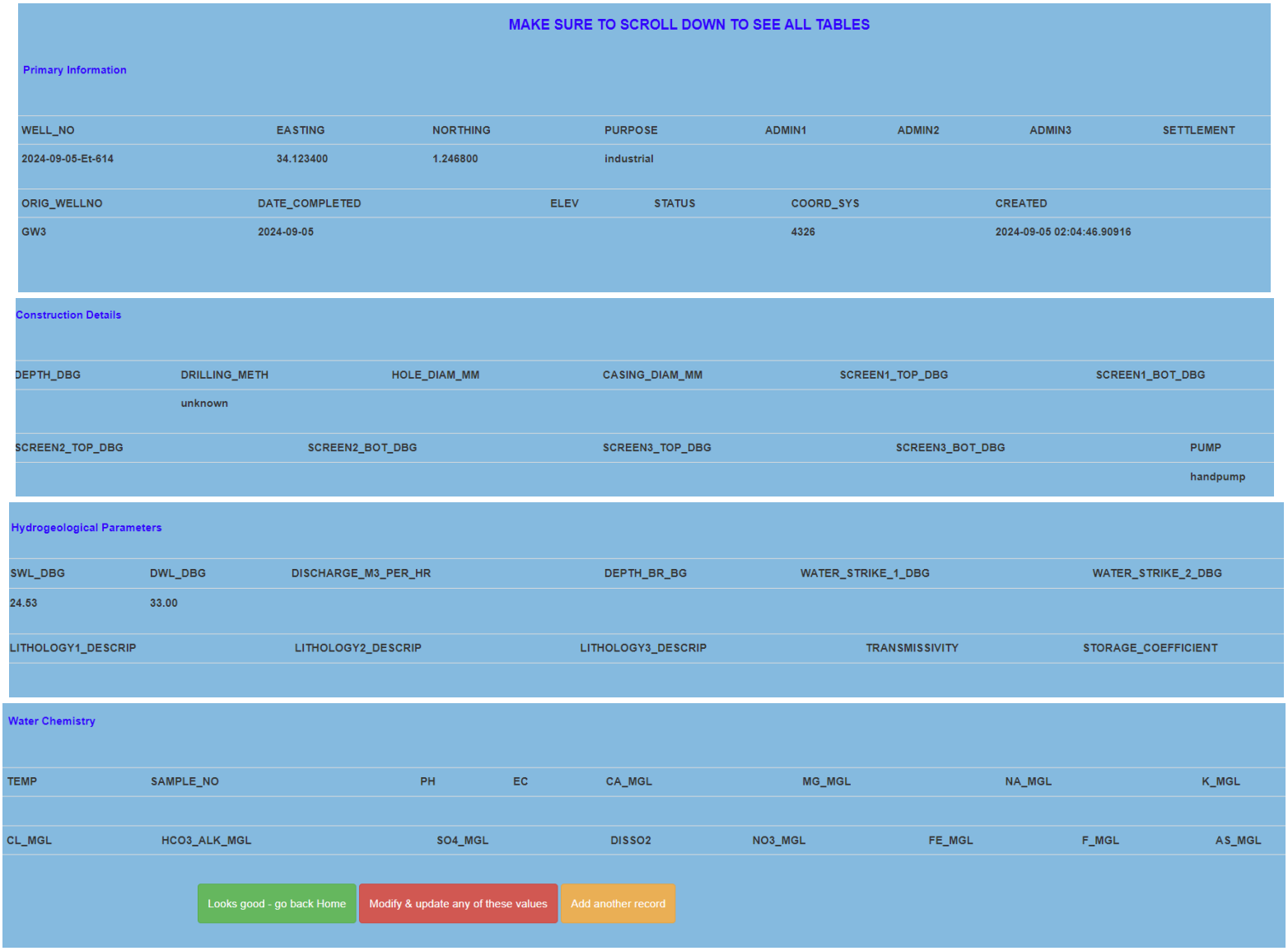

Add Record
Chloride mg/l
Chloride mg/l
Bicarbonate as Alkalinity
Bicarbonate as Alkalinity
Sulfate mg/l
Sulfate mg/l
Dissolved Oxygen
Dissolved Oxygen
Nitrate mg/l
Nitrate mg/l
Iron Fe+2 mg/l
Iron Fe+2 mg/l
Fluoride mg/l
Fluoride mg/l
Arsenic mg/l
Arsenic mg/l

Temperature should be in degrees Celsius, EC in uS/cm. chemical parameters are in mg/l which are converted tom milliequivalents per liter by the database.

This form also has constraints. ***Like all the numeric inputs on all the forms negative numbers are not allowed nor are symbols such as < , >, or text such NA***.

This form has maximum values restrictions as discussed earlier in this manual.

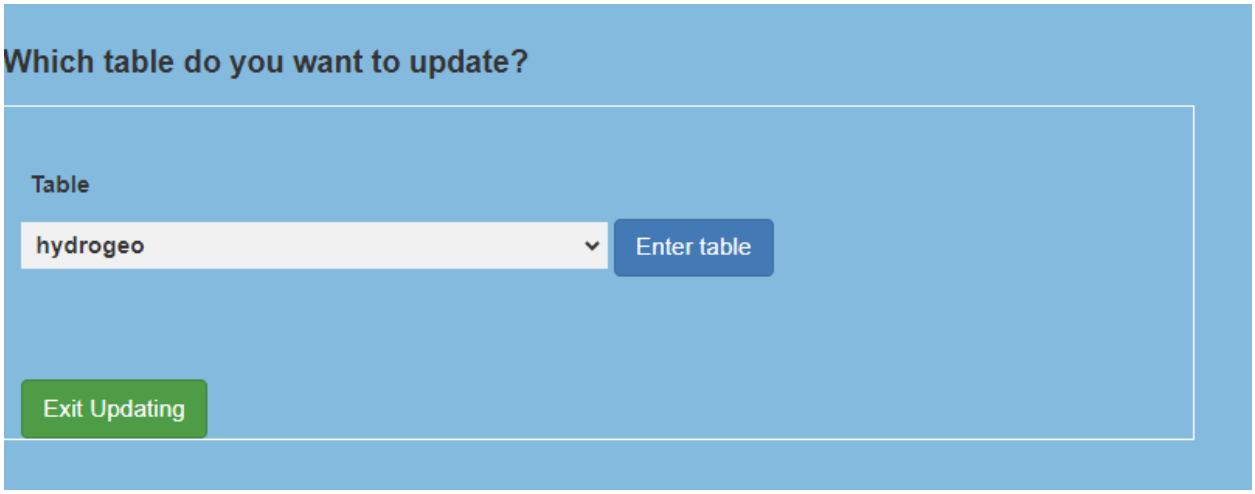
Click the 'Add Record' button – this will bring you to a review page to examine the information you just put in for all the forms. Make sure to scroll down to see the whole page.



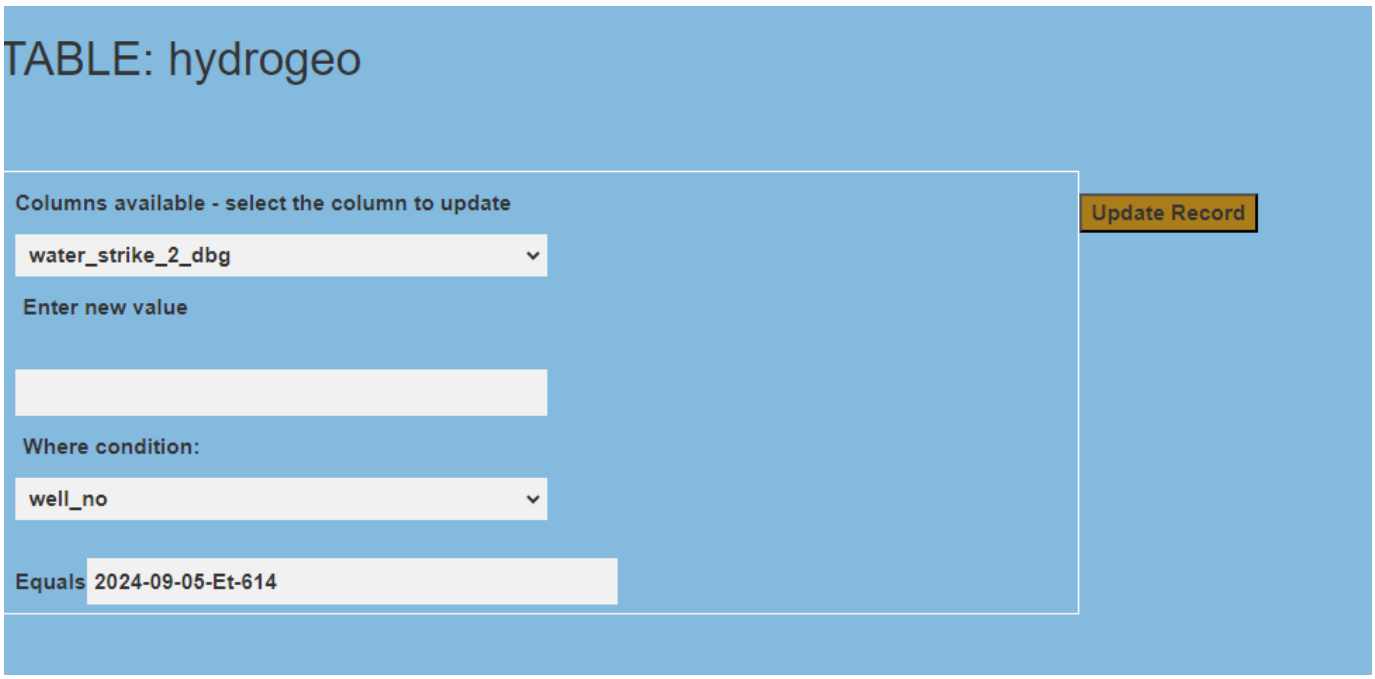
The buttons offer three choices – leave editing and go back home, modify the information you just entered, or add another set of well records.

Updating records using the web forms

If after finishing entering a well record, and you press the red button, the screen appears to choose the table you want to update.



Once you Enter table the next form appears with a dropdown list of columns in the table – choose the one to update and put in the new value. Notice that the Where condition is already filled in and cannot be changed. This is the well_no of the well the user was just working on.



Click the Update record button.

You can update pre-existing wells in the database by clicking on the menu item Update Well info in the navigation bar



Here, there is an opportunity to update more tales than just wells, such as river gauge and hydromet stations. A WEHERE CONDITION COLUMN MUST BE SUPPLIED or the form won't process the request. This is because without a Where condition, it is possible to update all records with the same value for that column for all the records in the table. A dropdown list provides possible columns to choose from. The value the column should equal to is also required to choose the correct record(s).

Note that there is no way to delete records using the browser. This feature was originally incorporated, but then removed. Deletion of records must be handled by the database administrator directly through the database. Because of the DELETE CASCADE action on the foreign well_no keys of the other tables, deleting a well in the well_parent table will delete the associated records in the other three child tables. Such decisions must be made carefully.

ANNEX 1:

PROGRAMMING CODE

General Procedure

There are three main varieties of code used by the application. The most primary is Structured Query language, SQL, which is the language of all database systems. Any action which requires reading from or writing to the database must first be formulated with an SQL query. Once this is done, it is the task of the server-side language, in this case PHP, to interpret the SQL to either respond to, or receive requests from the browser. However, the only scripting language the browser understands is JavaScript (CSS and HTML are considered markup languages for presentation, not actionable scripts). There are many tricks to translating between PHP and JavaScript, a common one being the PHP functions json_decode and json_encode functions, but an often-used alternative for complex behavior is the use of AJAX to call the PHP scripts. The explanation of AJAX as a hybrid technology that uses the XMLHttpRequest (XHR) object is beyond the scope of this document but suffice to say it can be called from within JavaScript, and therefore, from the browser, and without reloading the whole page to get the server response.

This annex will highlight the most salient portions of scripts used to drive the application, starting from the bottom up, i.e. from database to server up to browser (and back in reverse order).

SQL

Structured Query language appears in much of the PHP code, which, combined, serves as the application's dialogue with the database. SQL statements will be seen surrounded by PHP code in the section on PHP coding, predominantly SELECT statements.

Triggers and other functions, such as f_make_pointgeom, already were covered in the Database Documentation section. Here, we concentrate on VIEWS, which form a major foundation for the graphs.

VIEWS

The following two views use the elevation values in the well_parent table to calculate elevations of other levels (screen elevations, depths, water strikes etc) in the construction and hydrogeo tables

```
CREATE OR REPLACE VIEW public.construction_elevation

 AS

 SELECT c.well_no,

    wp.elev,

    c.depth_dbg,

    wp.elev - c.depth_dbg AS "elevation well bottom",

    c.screen1_top_dbg AS "screen 1 top",
```

```
    wp.elev - c.screen1_top_dbg AS "elevation screen 1 top",
    c.screen1_bot_dbg AS "screen 1 bottom",
    wp.elev - c.screen1_bot_dbg AS "elevation screen 1 bottom",
    c.screen2_top_dbg AS "screen 2 top",
    wp.elev - c.screen2_top_dbg AS "elevation screen 2 top",
    c.screen2_bot_dbg AS "screen 2 bottom",
    wp.elev - c.screen2_bot_dbg AS "elevation screen 2 bottom",
    c.screen3_top_dbg AS "screen 3 top",
    wp.elev - c.screen3_top_dbg AS "elevation screen 3 top",
    c.screen3_bot_dbg AS "screen 3 bottom",
    wp.elev - c.screen3_bot_dbg AS "elevation screen 3 bottom"
   FROM well_parent wp
     JOIN construction c ON c.well_no::text = wp.well_no::text;

ALTER TABLE public.construction_elevation
    OWNER TO postgres;
COMMENT ON VIEW public.construction_elevation
    IS 'calculates elevation of all screens in construction table from the elevation in parent well table';

CREATE OR REPLACE VIEW public.hydrogeo_elevation
 AS
 SELECT h.well_no,
    wp.elev,
    h.swl_dbg AS swl,
    wp.elev - h.swl_dbg AS "elevation swl",
    h.dwl_dbg AS dwl,
    wp.elev - h.dwl_dbg AS "elevation dwl",
    h.depth_br_bg AS "depth to bedrock",
    wp.elev - h.depth_br_bg AS "elevation of bedrock",
```

```
    h.water_strike_1_dbg AS "first water strike",
    wp.elev - h.water_strike_1_dbg AS "elevation first strike",
    h.water_strike_2_dbg AS "second water strike",
    wp.elev - h.water_strike_2_dbg AS "elevation second water strike"
   FROM well_parent wp
     JOIN hydrogeo h ON h.well_no::text = wp.well_no::text;

ALTER TABLE public.hydrogeo_elevation
    OWNER TO postgres;
COMMENT ON VIEW public.hydrogeo_elevation
    IS 'calculates elevation of water levels, depth to bedrock, and water strikes from elevation in parent well table';
```

Another view was used to make interpolation of variables in the web application easier to code

```
CREATE OR REPLACE VIEW public.interp1
 AS
 SELECT wp.well_no,
    wp.orig_wellno,
    wp.geom,
    c.depth_dbg,
    h.water_strike_1_dbg,
    h.discharge_m3_per_hr,
    h.transmissivity,
    h.storage_coefficient,
    wq.ca_meql,
    wq.mg_meql,
    wq.na_meql,
    wq.k_meql,
    wq.cl_meql,
```

```
    wq.so4_meql
   FROM well_parent wp
     LEFT JOIN construction c ON c.well_no::text = wp.well_no::text
     LEFT JOIN hydrogeo h ON h.well_no::text = wp.well_no::text
     LEFT JOIN wq ON wq.well_no::text = wp.well_no::text;

ALTER TABLE public.interp1
    OWNER TO postgres;
```

A special view was created for the water chemistry (quality) table, **wq**. This was necessary was because of the all the generated columns used for conversion for mgl to meql. Triggers don't work with generated columns, so in order to compute parameters such as total cations, , total dissolved ions, and analytical error, a view had to be created. Following best practice in database design a view is preferred rather than storing data that could be derived from the primary data. This could also be said of the generated columns, so they also may be removed, and the calculations put into a view. A view does not store data, it is merely a SQL statement that produces output on the fly. In any case, the view that calculates the sums, tdi, and percent error is formulated as follows:

```
CREATE OR REPLACE VIEW public.wq_error
 AS
 WITH wqsums AS (
         SELECT wq.well_no,
            wq.sample_no,
            wq.ca_meql + wq.mg_meql + wq.na_meql + wq.k_meql AS sumcat,
            wq.cl_meql + wq.hco3_alk_meql + wq.so4_meql AS suman,
            wq.ca_meql + wq.mg_meql + wq.na_meql + wq.k_meql + wq.cl_meql + wq.hco3_alk_meql + wq.so4_meql AS
tdi
           FROM wq
        )
 SELECT well_no,
    sample_no,
    sumcat AS "total cations",
```

```
 suman AS "total anions",
 tdi,
 (sumcat- suman) / tdi * 100::numeric AS "percent error"
FROM wqsums;
```

This gives the result:

well_no character varying (50)	sample_no character varying (50)	total cations numeric	total anions numeric	tdi numeric	percent error numeric
2024-04-08-Et-929		8.589879	2.489572	11.079451	55.05965051878473039900
2024-04-08-Et-441		8.589879	2.489572	11.079451	55.05965051878473039900
2024-04-08-Et-841		8.589879	2.489572	11.079451	55.05965051878473039900
2024-04-08-Et-812		8.589879	2.489572	11.079451	55.05965051878473039900
2024-04-08-Et-750		27.760194	7.162611	34.922805	58.98032245691604669200
2024-04-08-Et-345	111	30.426033	2.489572	32.915605	84.87299868861593156200
2024-04-08-Et-421		30.426033	7.162611	37.588644	61.88949513581814763000

Fake random data was used, so hopefully with real data, we won't get such horrendous analytical errors.

PHP

ADMIN MODULE

File: init.php

Every module has an includes folder, which mainly holds PHP files needed by other PHP files. This avoids repeating the same lines of code in every file that needs it. The most important of these 'include' PHP files is init.php, without which the application would not run at all, since it makes the database connection

```
<?php
ob_start();
session_start();
$host = '127.0.0.1';
```

```
$dbname = 'gwis';
$charset = 'utf8mb4';
$dsn = "pgsql:host=$host; dbname=$dbname; port =5432;";
$options = [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES => false
];


$pdo = new PDO($dsn, 'postgres', 'f1234u5678', $options);


$root_directory = "html";
$from_email = " ougas@icpac.net ";
$reply_email = " ougas@icpac.net ";
include "php_functions.php";
```

The first two lines initialize a session buffer and start a session. This is necessary since session variables are used to track the user that has logged in and what country he/she is from. The other lines carry the connection configuration variables: database driver, host, database name – and are encapsulated in a PDO object which also has user name and password as parameters. Also note that ini.php itself includes another file, php_functions.php

File : php_functions.php

This file, also in the includes directory, contains numerous functions, mainly used in the admin module.

The functions are described in order of appearance.

The task of transferring browser-based form values into arrays suitable for insertion as database records posed a problem that caused a dealy in development for nmore than a week. The problem could probably have been overcome using jQuery AJAX with the serialize method (or perhaps not!) . the traditional approach of using PHP associative arrays suffers from the drawback of not being able to handle NULL values, which is a requirement when filling in well information. A solution was finally found from an expert PHP developer on GitHub.

```
/* Converts blank strings in array to null values.

For instance, can use on $_POST to convert empty form inputs or inputs
with only spaces to null for database insert.

*/



function convert_blank_strings_to_null(array $array, $recursive =
false)

{

    $newArray = array();

    foreach ($array as $key => $value) {

        if (is_string($value)) {

            if (trim($value) == '') {

                $newArray[$key] = null;

            } else {

                $newArray[$key] = $value;

            }

        } elseif (is_array($value)  &&  $recursive == true) {

            $newArray[$key] = convert_blank_strings_to_null($value);

        } else {

            $newArray[$key] = $value;

        }

    }

    return $newArray;

}
```

Without the above code, it would have been impossible to transfer well information from the form into the database due to errors in the associative arrays.

This function redirects the brower to another url

```
function redirect($loc)
```

```
{
    header("Location: {$loc}");
}
```

This function generates the token used to activate the user by generating a hashed version of time in milliseconds coupled with a random number

```
function generate_token()
{
    return md5(microtime() . mt_rand());
}
```

This determines login status. Note the consultant decided not to use the COOKIE functionality, so “Remember me” is not possible, requiring the user to login every time to access the application. The consultant forgot to remove that part of the code after changing his mind, so it is possible to reinstate this in the future.

```
function logged_in()
{
    if (isset($_SESSION['username'])) {
        return true;
    } else {
        if (isset($_COOKIE['username'])) {
            $_SESSION['username'] = $_COOKIE['username'];
            return true;
        } else {
            return false;
        }
    }
}
```

The following code sends messages due to log in errors and sets the style for the messages.

```
function set_msg($msg, $level = 'danger')

{

    if (($level != 'primary') && ($level != 'success') && ($level != 
'info') && ($level != 'warning')) {

        $level = 'danger';

    }

    if (empty($msg)) {

        unset($_SESSION['message']);

    } else {

        $_SESSION['message'] = "<h5 class='bg-{$level} text-
center'>{$msg}</h5>";

    }

}


function show_msg()

{

    if (isset($_SESSION['message'])) {

        echo $_SESSION['message'];

        unset($_SESSION['message']);

    }

}
```

The next function, written for the native PHP sendmail function, is not used as it has been decided that the third party library PHP Mailer be used instead because of increased functionality and security.

```
function send_mail($to, $subject, $body, $from, $reply)

{

    $headers = "From: {$from}" . "\r\n" . "Reply-To: {$reply} " . " 
\r\n " . "X-Mailer: PHP/" . phpversion();
```

```
    if ($_SERVER['SERVER_NAME'] != "localhost") {
        mail($to, $subject, $body, $headers);
        set_msg("Email sent to '{$to}'. Please check email to activate
your account");
        redirect('index.php');
    } else {
        echo "<hr><p>To: {$to}</p><p>Subject:
{$subject}</p><p>{$body}</p><p>" . $headers . "</p><hr>";
    }
}
```

These next two functions are also not used – they were written to check for valid institutional domains, but after being advised that authorized users would not being using institutional emails, the functions have been commented out.

```
function validate_email($email)
{
    //$email = filter_var($email, FILTER_SANITIZE_EMAIL);
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        return false;
    }
}

function valid_org($email, $org)
{
    if ($org < 100) {
        $valid_domain = ['igad', 'icpac', 'gov'];
        /*if( preg_match("/^( [a-zA-Z0-9\. -]+)@$domain). ([a-
z]{2,20}) (.[a-z]{2,20} )?$/"))*/
        $domain = strtok($email, "@");
```

```
        $domain = strtok("@");  //run a second time without the string
arg
        $domain = strtok($domain, ".");  //run a third time to extract
between @ and the dot


        if (in_array($domain, $valid_domain)) {


            echo "You are authorized to edit records for
{$_SESSION['country']}";
            return true;
        } else {
            echo "Your email domain is not authorized to edit data.
Use organization email";
            return false;
        }
    } else {


        echo "you can view data but not edit it";
        return false;
    }
}
```

Database functions in the php_functions.php file – these make calls to the database, most of them that involve user input utilizing **prepare** statements to prevent SQL injection They also use try-catch blocks to intercept errors and display the relevant error message.

```
//   ******************  DATABASE FUNCTIONS
*******************************


function count_field_val($pdo, $tbl, $fld, $val)
{
```

```
    try {
        $sql = "SELECT {$fld} FROM {$tbl} WHERE {$fld}=:value";
        $stmnt = $pdo->prepare($sql);
        $stmnt->execute([':value' => $val]);
        return $stmnt->rowCount();
    } catch (PDOException $e) {
        return $e->getMessage();
    }
}

function return_field_data($pdo, $tbl, $fld, $val)
{
    try {
        $sql = "SELECT * FROM {$tbl} WHERE {$fld}=:value";
        $stmnt = $pdo->prepare($sql);
        $stmnt->execute([':value' => $val]);
        return $stmnt->fetch();
    } catch (PDOException $e) {
        return $e->getMessage();
    }
}

function get_validationcode($user, $pdo)
{
    try {
        $stmnt = $pdo->prepare("SELECT validationcode FROM users WHERE
username=:username");
        $stmnt->execute([':username' => $user]);
        $row = $stmnt->fetch();
        return $row['validationcode'];
    } catch (PDOException $e) {
```

```
        return $e->getMessage();
    }
}

function get_orgcode($user, $pdo)
{
    try {
        //$username=$_SESSION['username'];
        $stmnt = $pdo->prepare("SELECT org_code FROM users WHERE
username=:username");
        $stmnt->execute([':username' => $user]);
        $row = $stmnt->fetch();
        return $row['org_code'];
    } catch (PDOException $e) {
        return $e->getMessage();
    }
}

function get_email($user, $pdo)
{
    try {
        //$username=$_SESSION['username'];
        $stmnt = $pdo->prepare("SELECT email FROM users WHERE
username=:username");
        $stmnt->execute([':username' => $user]);
        $row = $stmnt->fetch();
        return $row['email'];
    } catch (PDOException $e) {
        return $e->getMessage();
    }
}
```

```
function get_country($user, $pdo)
{
    try {
        $stmnt = $pdo->prepare("SELECT country FROM users WHERE
username=:username");
        $stmnt->execute([':username' => $user]);
        $row = $stmnt->fetch();
        return $row['country'];
    } catch (PDOException $e) {
        return $e->getMessage();
    }
}


/*THIS FUNCTION NOT USED AT THIS TIME
function create_role($pdo, $uname, $pword)
{
    try {
        $sql_role = "create role {$uname} login password '{$pword}'";
        $stmt = $pdo->query($sql_role);
    } catch (PDOException $e) {
        echo $e->getMessage();
    }
}

    /*function get_srid($srid) {
        if (!is_null($easting) || !is_null($northing))
            echo "<p> For coordinates you must enter a coordinate
system/p>";
```

```
            try {
                $result=$pdo->query("SELECT * FROM crs_lookup ORDER BY
crs desc");
                 foreach ($result as $row) {
                 echo "<option
value={$row['srid']}>{$row['crs']}</option>";

                    }

                } catch(PDOException $e) {
                          echo "Error: ".$e->getMessage();
                     }

    }*/
```

EMBEDDED PHP – Much of the PHP code in the application is embedded inside the same file that holds HTML, CSS, and JavaScript (in other words, the web pages that are presented to the user). In most cases, the PHP code appears at the top of the file, except in cases of SELECT HTML elements where values are retrieved or sent to the database via dropdown lists.

Embedded PHP - File: register.php

This code takes info from the registration form, validates the input fields including minimum length of 6 characters for username and password, puts the user information into the users table in the database, generates the token needed for activation, and sends error messages if appropriate. Finally, it sends the activation email with the token link, receives the response when the user clicks on the link in the email, and redirects the user to the login page.

```
<?php include "includes/init.php" ?>
<?php //include "includes/php_functions.php"
?>
```

```php
<?php
if ($_SERVER['REQUEST_METHOD'] == "POST") {
    $fname = $_POST['fullname'];
    $uname = $_POST['username'];
    $email = $_POST['email'];
    $email_conf = $_POST['email_confirm'];
    $pword = $_POST['password'];
    $pword_conf = $_POST['password_confirm'];
    $org = $_POST['org'];
    $country = $_POST['country'];
    $_SESSION['country'] = $country;

    if (strlen($uname) < 6) {
        $error[] = "User name must be at least 6 characters long";
    }
    if (strlen($pword) < 6) {
        $error[] = "Password must be at least 6 characters long";
    }
    if ($pword != $pword_conf) {
        $error[] = "Passwords do not match";
    }
    if ($email != $email_conf) {
        $error[] = "Email addresses do not match";
    }
    if (count_field_val($pdo, "users", "username", $uname) != 0) {
        $error[] = "Username '{$uname}' already exists";
    }
    if (count_field_val($pdo, "users", "email", $email) != 0) {
        $error[] = "Email '{$email}' already exists";
    }
```

```
    /*if (!validate_email(($_POST['email']))) {
        $error[] = "Email '{$email}' not valid -  no dot or domain";
    }*/

    if (!isset($error)) {

        $vcode = generate_token();
        try {
            $sql = "INSERT INTO users (fullname, username, email,
password,  validationcode,  joined, last_login, org_code, country)
                VALUES (:fullname,  :username, :email, :password,
:vcode,  current_date, current_date, :org_code, :country)";
            $stmnt = $pdo->prepare($sql);
            $user_data = [':fullname' => $fname, ':username' =>
$uname, ':email' => $email, ':password' => password_hash($pword,
PASSWORD_BCRYPT),  ':vcode' => $vcode, ':org_code' => $org, ':country'
=> $country];
            $stmnt->execute($user_data);



            //redirect('index.php');
            $body = "Please go to
http://{$_SERVER['SERVER_NAME']}/{$root_directory}/activate.php?user={
$uname}&code={$vcode} in order to activate account";
            send_mail($email, "Activate User", $body, $from_email,
$reply_email);

            //grant($uname, $organization);



        } catch (PDOException $e) {
            echo "Error: " . $e->getMessage();
```

```
        }
    }
} else {
    $fname = "";
    $uname = "";
    $email = "";
    $email_conf = "";
    $pword = "";
    $pword_conf = "";
    $org = "";
    $country = "";
}
?>
```

File: activate.php

This a pure PHP file that is the target of the link in the activation email. It receives the token through a GET request and checks the generated token in the database, and if a match is found, the activation column is set to '1' (true) in the users table and thus activates the user. This verifies that the email address the user submitted in the registration form is valid and correct.

```
<?php include "includes/init.php" ?>
<?php
    if (isset($_GET['user'])) {
        $user = $_GET['user'];
        if (isset($_GET['code'])) {
            $code = $_GET['code'];
            $db_code = get_validationcode($user, $pdo);
            if ($code == $db_code) {
                try {
                    $stmnt=$pdo->prepare("UPDATE users SET active=1
WHERE username=:username");
                    $stmnt->execute([':username'=>$user]);
```

```
                    set_msg("User Activated, Please login to access
the GWIS", "success");
                    redirect('index.php');
                } catch(PDOException $e) {
                    echo "Error: {$e}";
                }
            } else {
                set_msg("Validation code does not match the
database");
                redirect('index.php');
            }
        } else {
            set_msg("No validation code included with activate
request");
            redirect('index.php');
        }
    } else {
        set_msg("No user included with activate request");
        redirect('index.php');
    }
?>
```

Embedded PHP - file: index.php

This is the login page. Code takes information from the web form, checks username and password, and sends error messages if values not found in database (not registered) or if not activated. It also puts the username and the country inside session variables.

```
<?php include "includes/init.php" ?>

<?php include "includes/header.php" ?>
```

```
<?php include "includes/nav.php" ?>


<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $username = $_POST['username'];
    $password = $_POST['password'];
    if (isset($_POST['remember'])) {
        $remember = "on";
    } else {
        $remember = "off";
    }



    if (count_field_val($pdo, "users", "username", $username) > 0) {
        $user_data = return_field_data($pdo, "users", "username",
$username);
        if ($user_data['active'] == 1) {
            if (password_verify($password, $user_data['password'])) {
                set_msg("Logged in successfully", "success");
                $_SESSION['username'] = $username;
                $country = get_country($username, $pdo);
                $_SESSION['country'] = $country;


                /*if ($remember="on") {
                        setcookie("username", $username,
time()+86400);
                    }*/
                redirect("entrance.php");
            } else {


                set_msg("Password is invalid");
```

```
            }


   } else {
            set_msg("User '{$username}' found but has not been
activated");
        }
    } else {


        set_msg("User '{$username}' does not exist");
    }
} else {
    $username = "";
    $password = "";
}
?>
```

Embedded PHP – file: entrance.php

This is the landing page (which needs to be filled with html CONTENT). The PHP script here calls a function in the php_functions.php file that basically puts the username into a session variable, checks the organization code in the database, and provides the appropriate navigation bar. Those that are authorized will see the **Edit data** item in the menu bar they are provided with. Those users not belonging to an organization authorized to edit, will get a menu bar that lacks that item.

```
<?php include "includes/init.php" ?>
<?php
    if (logged_in()) {
        $username=$_SESSION['username'];
    } else {
        set_msg("Please log-in and try again");
        redirect('index.php');
    }
```

```
          //$org = get_orgcode($username,$pdo);
          //$country = get_country($username,$pdo);
          $email = get_email($username, $pdo);
         //$_SESSION['country'] = get_country($username,$pdo);

         $nav_select =get_orgcode($username,$pdo);
          //$valid_domain = valid_org($email,$org);

          if ($nav_select < 100 )//&& $valid_domain) {
             {
             include "includes/navigation3.php";


         }else{

             include "includes/navigation2.php";
        }




?>
```

Embedded PHP – File: reset1.php

This the PHP behind the reset1 form which asks for the username, then looks up the email to send a set password link

```
<?php include "includes/init.php"; ?>
<?php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $username = $_POST['username'];
```

```
    if (count_field_val($pdo, "users", "username", $username) > 0) {
        $row = return_field_data($pdo, "users", "username",
$username);
        $body = "Please go to
http://{$_SERVER['SERVER_NAME']}/{$root_directory}/reset_2.php?user={$
username}&code={$row['validationcode']} in order to reset your
password";
        send_mail($row['email'], "Reset Password", $body, $from_email,
$reply_email);
    } else {
        set_msg("User '{$username}' was not found in the database");
    }
}
?>
```

Embedded PHP – File: reset2.php

This the PHP behind the reset2 form which receives the link from the email which includes the user's validation code, then looks up the validation code to verify the user. If the user is verified, the new password information in the form provided is inserted into the users table for that user using an UPDATE statement.

```
<?php include "includes/init.php" ?>
<?php
    if ($_GET['user']) {
        if ($_GET['code']) {
            $username=$_GET['user'];
            $vcode=$_GET['code'];
            if (count_field_val($pdo, "users", "username",
$username)>0) {
                $row=return_field_data($pdo, "users", "username",
$username);
                if ($vcode!=$row['validationcode']) {
                    set_msg("Validation code does not match
database");
                    redirect("index.php");
                }
```

```
            } else {
                set_msg("User '{$username}' not found in database");
                redirect("index.php");
            }
        } else {
            set_msg("No validation code included with reset request");
            redirect("index.php");
        }
    } else {
        set_msg("No user included with reset request");
        redirect("index.php");
    }
    if ($_SERVER['REQUEST_METHOD']=="POST") {
        try {
            $password=$_POST['password'];
            $pword_confirm=$_POST['password_confirm'];
            if ($password==$pword_confirm) {
                $stmnt=$pdo->prepare("UPDATE users SET
password=:password WHERE username=:username");
                $user_data=[':password'=>password_hash($password,
PASSWORD_BCRYPT), ':username'=>$username];
                $stmnt->execute($user_data);
                set_msg("Password successfully updated. Please log
in");
                redirect("index.php");
            } else {
                set_msg("Passwords don't match");
            }
        } catch(PDOException $e) {
            echo "Error: ".$e->getMessage();
        }
```

```
    }
?>
```

File: logout.php

This is a pure PHP file that unsets the session variables and destroys the session. User is redirected out of the GWIS.

```
<?php
    include "includes/init.php";
    unset($_SESSION['username']);
    unset($_SESSION['country']);
    unset($_SESSION['elev']);
    unset($_SESSION['well_no']);
    if (isset($_COOKIE['username'])) {
        setcookie('username', 'delete', time()-3600);
    }
    redirect('goodbye.php');
?>
```

Embedded PHP – record by record input forms for the database tables well_parent, construction, hydrogeo, and wq – wellparent_add.php, well_construct.php, hydrogeo.php, and wq.php respectively. All these files are of similar structure, PHP code embedded in HTML forms that insert values into the above-mentioned tables through the use of POST variables and prepared statements, and so are presented together. Here it is also worth noting that well_no is kept in a session variable, and this same value is inserted into the other three tables automatically (since it is the same well). When the add button is clicked, the page redirects to the next form in the sequence.

File: wellparent_add.php

```
<?php include "includes/init.php" ?>
```

```
<?php
if ($_SERVER['REQUEST_METHOD'] == "POST") {
```

```
    $easting = $_POST['easting'];


    $northing = $_POST['northing'];


    $purpose = $_POST['purpose'];
    $country = $_SESSION['country'];
    $admin1 = $_POST['admin1'];
    $admin2 = $_POST['admin2'];
    $admin3 = $_POST['admin3'];
    $settlement = $_POST['settlement'];
    $orig_wellno = $_POST['orig_wellno'];


    $date_completed = $_POST['date_completed'];
    $coord_sys = isset($_POST['coord_sys']) ? $_POST['coord_sys'] :
null;
    $srid = $_POST['srid'];
    $elev = $_POST['elev'];
    $status = $_POST['status'];
    $well_no = $date_completed . "-" . substr($country, 0, 2) . "-" .
mt_rand(1, 9) . mt_rand(10, 50);
    $_SESSION['well_no'] = $well_no;
   // $_SESSION['elev'] = $elev;




    $new_post = convert_blank_strings_to_null($_POST, $recursive =
false);


    $easting = $new_post['easting'];
    $northing = $new_post['northing'];
```

```
    $purpose = $new_post['purpose'];
    $admin1 = $new_post['admin1'];
    $admin2 = $new_post['admin2'];
    $admin3 = $new_post['admin3'];
    $settlement = $new_post['settlement'];
    $orig_wellno = $new_post['orig_wellno'];
    $coord_sys = $new_post['coord_sys'];
    $srid = $new_post['srid'];
    $elev = $new_post['elev'];
    $status = $new_post['status'];



    try {


        $sql = "INSERT INTO well_parent (well_no, easting, northing,
purpose, country,  admin1,  admin2,
            admin3, settlement, orig_wellno,  date_completed,
coord_sys, elev, status)
            VALUES (:well_no, :easting, :northing, :purpose, :country,
:admin1,  :admin2, :admin3,
            :settlement, :orig_wellno, :date_completed, :coord_sys,
:elev, :status)";
        $stmnt = $pdo->prepare($sql);
        $user_data = [
            ':well_no' => $well_no, ':easting' => $easting,
':northing' => $northing, ':purpose' => $purpose,  ':country' =>
$country,
            ':admin1' => $admin1, ':admin2' => $admin2, ':admin3' =>
$admin3, ':settlement' => $settlement, ':orig_wellno' => $orig_wellno,
            ':date_completed' => $date_completed, ':coord_sys' =>
$coord_sys, ':elev' => $elev, ':status' => $status
        ];
```

```
        $stmnt->execute($user_data);


        /*echo "<p> <b>user_data array passed to stmnt-
>execute</b></p>";
            echo '<pre>'; print_r($user_data); echo '</pre>';*/
        echo $_SESSION['country'];
        echo $sql;







        //create_role( $pdo, $uname, $pword);*/


        redirect('well_construct.php');
    } catch (PDOException $e) {


        //echo $_SESSION['country'];



        echo "Error: " . $e->getMessage();
    }
} else {
    $well_no = "";
    $easting = "";
    $northing = "";
    $purpose = "";
    //$country ="";
    $admin1 = "";
    $admin2 = "";
    $admin3 = "";
    $settlement = "";
```

```
    $orig_wellno = "";
    $contractor = "";
    $date_completed = "";
    $coord_sys = "";
    $srid = "";
    $elev = "";
    $status = "";
}
```

Also note the embedded PHP in the HTML SELECT elements to produce drop down lists for ‘purpose’ ‘coordinate system’.

```
<select name='purpose' id='purpose' tabindex="3" required>
                <?php
                try {
                    $result = $pdo->query("SELECT purpose FROM purpose
ORDER BY purpose");
                    foreach ($result as $row) {
                        echo "<option
value={$row['purpose']}>{$row['purpose']}</option>";
                    }
                } catch (PDOException $e) {
                    echo "Error: " . $e->getMessage();
                }
                ?>
            </select>


<select name="coord_sys" id="coord_sys" class='form-control'>
                <?php
                if (!is_null($easting) || !is_null($northing))
                    echo "<p> For coordinates you must enter a
coordinate system/p>";
                try {
```

```
                    $result = $pdo->query("SELECT * FROM crs_lookup
where countries like '%{$_SESSION['country']}%' OR countries like
'Entire%' ORDER BY system desc");
                    foreach ($result as $row) {
                        echo "<option
value={$row['srid']}>{$row['system']}</option>";
                    }
                } catch (PDOException $e) {
                    echo "Error: " . $e->getMessage();
                }
                ?>
            </select>
```

File: well_construct.PHP

```
<?php include "includes/init.php" ?>



<?php print_r($_SESSION);
$well_no = $_SESSION['well_no'];

//$elev = $_SESSION['elev'];
//echo $elev;

$country = $_SESSION['country'];
?>
<?php
if ($_SERVER['REQUEST_METHOD'] == "POST") {
    //$well_no = $_POST['well_no'];
    $depth_dbg = $_POST['depth_dbg'];
    $drilling_meth = $_POST['drilling_meth'];
    $hole_diam_mm = $_POST['hole_diam_mm'];
```

```
    $casing_diam_mm = $_POST['casing_diam_mm'];
    $screen1_top_dbg = $_POST['screen1_top_dbg'];
    $screen1_bot_dbg = $_POST['screen1_bot_dbg'];
    $screen2_top_dbg = $_POST['screen2_top_dbg'];
    $screen2_bot_dbg = $_POST['screen2_bot_dbg'];
    $screen3_top_dbg = $_POST['screen3_top_dbg'];
    $screen3_bot_dbg = $_POST['screen3_bot_dbg'];
    $pump = $_POST['pump'];

    $new_post = convert_blank_strings_to_null($_POST, $recursive =
false);
    $depth_dbg = $new_post['depth_dbg'];
    $drilling_meth = $new_post['drilling_meth'];
    $hole_diam_mm = $new_post['hole_diam_mm'];
    $casing_diam_mm = $new_post['casing_diam_mm'];
    $screen1_top_dbg = $new_post['screen1_top_dbg'];
    $screen1_bot_dbg = $new_post['screen1_bot_dbg'];
    $screen2_top_dbg = $new_post['screen2_top_dbg'];
    $screen2_bot_dbg = $new_post['screen2_bot_dbg'];
    $screen3_top_dbg = $new_post['screen3_top_dbg'];
    $screen3_bot_dbg = $new_post['screen3_bot_dbg'];
    $pump = $new_post['pump'];
    $new_session = convert_blank_strings_to_null($_SESSION, $recursive
= false);
    //$elev = $new_session['elev'];

    //$_SESSION['well_no'] = $well_no;

    //$_SESSION['elev'] = $elev;

    try {
```

```
        $sql = "INSERT INTO construction (well_no,  depth_dbg,
drilling_meth, hole_diam_mm,  casing_diam_mm,  screen1_top_dbg ,
screen1_bot_dbg,

            screen2_top_dbg , screen2_bot_dbg , screen3_top_dbg,
screen3_bot_dbg, pump)

            VALUES (:well_no,  :depth_dbg, :drilling_meth,
:hole_diam_mm,  :casing_diam_mm,  :screen1_top_dbg , :screen1_bot_dbg,

            :screen2_top_dbg , :screen2_bot_dbg , :screen3_top_dbg,
:screen3_bot_dbg, :pump)";

        $stmnt = $pdo->prepare($sql);

        $user_data = [

            ':well_no' => $well_no, ':depth_dbg' => $depth_dbg,
':drilling_meth' => $drilling_meth,  ':hole_diam_mm' => $hole_diam_mm,

            ':casing_diam_mm' => $casing_diam_mm, ':screen1_top_dbg'
=> $screen1_top_dbg, ':screen1_bot_dbg' => $screen1_bot_dbg,
':screen2_top_dbg' => $screen2_top_dbg,

            ':screen2_bot_dbg' => $screen2_bot_dbg,
':screen3_top_dbg' => $screen3_top_dbg, ':screen3_top_dbg' =>
$screen3_top_dbg, ':screen3_top_dbg' => $screen3_top_dbg,

            ':screen3_bot_dbg' => $screen3_bot_dbg, ':pump' => $pump

        ];

        //print_r($user_data);


        $stmnt->execute($user_data);




        redirect('hydrogeo.php');

    } catch (PDOException $e) {

        echo "Error: " . $e->getMessage();

    }

} else {


    $depth_dbg = "";

    $drilling_meth = "";
```

```
    $hole_diam_mm  = "";
    $casing_diam_mm = "";
    $screen1_top_dbg = "";
    $screen1_bot_dbg = "";
    $screen2_top_dbg = "";
    $screen2_bot_dbg = "";
    $screen3_top_dbg = "";
    $screen3_bot_dbg = "";
    $pump = "";
}


?>
```

Also note the embedded PHP in the HTML SELECT elements to produce drop down lists for ‘drilling method’ and ‘pump’.

```
<select name='drilling_meth' id='drilling_meth'>
                    <?php
                    try {
                        $result = $pdo->query("SELECT method FROM
drilling_meth ORDER BY id");
                        foreach ($result as $row) {
                            echo "<option
value={$row['method']}>{$row['method']}</option>";
                        }
                    } catch (PDOException $e) {
                        echo "Error: " . $e->getMessage();
                    }
                    ?>
                </select>

<select name='pump' id='pump'>
                <?php
```

```
                try {
                    $result = $pdo->query("SELECT pump_type FROM pump
ORDER BY pump_type");
                    foreach ($result as $row) {
                        echo "<option
value={$row['pump_type']}>{$row['pump_type']}</option>";
                    }
                } catch (PDOException $e) {
                    echo "Error: " . $e->getMessage();
                }
                ?>
            </select>
```

Graphs and Map Modules

These modules also have *include* folders with the same PHP files inside. The main idea behind this duplication is to have each module self-contained with its own resource folders. The main purpose is to either retrieve information from the database to feed dropdown lists and to send responses based on user input to bring back data relevant for graphs and maps.

The most important PHP file in the map module is gjson_load.php. This script is called by AJAX whenever a database table needs to be mapped. It goes record by record and field by field to construct a geoJSON object, which is required by Leaflet to place the records on the map.

```
<?php
    include "includes/init.php";
    $result = $pdo->query("SELECT *, ST_AsGeoJSON(geom, 5) AS geojson
FROM interp1");
    $features=[];
    foreach($result AS $row) {
        unset($row['geom']);
        $geometry=$row['geojson']=json_decode($row['geojson']);
        unset($row['geojson']);
        $feature=["type"=>"Feature", "geometry"=>$geometry,
"properties"=>$row];
```

```
        array_push($features, $feature);
    }
    $featureCollection=["type"=>"FeatureCollection",
"features"=>$features];
    echo json_encode($featureCollection);


?>
```

Often times, the database tables that are referred to are VIEWS (refer to the SQL section). Below is the code from the aquifer_hydro.php file in the Graphs module that accomplishes these tasks – receiving the variable for aquifer and returning the time series for that aquifer which is formed by aggragate functions within a joined view.:

```
<?php include "includes/init.php" ?>
<?php include "includes/navigation2.php"?>


<?php if($_SERVER['REQUEST_METHOD'] == "POST") {
    $aquifer = $_POST['aquifer'];
    //$aquifer_name = $_POST['aquifer_name'];
} else {
    $aquifer = "";
}


try {
    $sql = "SELECT * FROM trans_ann_climate tac inner join
trans_aquifers ta on ta.code=tac.code
    where year between '2000'  and '2023'and ta.name = '{$aquifer}';";
    $result = $pdo->query($sql);
    //if ($result->rowCount() > 0) {
    foreach ($result as $row) {
        $rainfall[] = $row['rainfall'];
        $et[] = $row['actual_et'];
        $year[] = $row['year'];
```

```
    }
} catch (PDOException $e) {
    echo $e->getMessage();
}


try {
    $sql2 = "SELECT tc.*, ta.name FROM trans_clime_monthly tc inner
join trans_aquifers ta on ta.code=tc.code where ta.name =
'{$aquifer}';";
    $result = $pdo->query($sql2);
    //if ($result->rowCount() > 0) {
    foreach ($result as $row) {
        $avg_rainfall[] = $row['avg_rainfall'];
        $avg_aet[] = $row['avg_aet'];
        $month[] = $row['month'];
    }
} catch (PDOException $e) {
    echo $e->getMessage();
}
```

And again, we see PHP being used for the SELECT dropdown list:

```
<select  name="aquifer" id="aquifer2" >
                        <option value="">--click here--</option>
                        <?php


                        try {
                            $result = $pdo->query("SELECT name, code
FROM trans_aquifers;");
                            foreach ($result as $row) {
                                echo "<option
value={$row['name']}>{$row['name']}</option>";
                            }
```

```
                        } catch (PDOException $e) {
                            echo "Error: " . $e->getMessage();
                        }
                        ?>
                    </select>
```

In timeseries.php, rows are retrieved from the mw-data table:

```
<?php include "includes/init.php"?>
<?php include "includes/navigation2.php"?>

<?php include "includes/header.php" ;



if($_SERVER['REQUEST_METHOD'] == "POST") {
    $mw_id = $_POST['mw_id'];
    //$aquifer_name = $_POST['aquifer_name'];
} else {
    $mw_id = "";
}
try {
    $sql = "SELECT distinct datetime::date, water_level_bg FROM
public.mw_data
    where mw_id = '{$mw_id}'
    order by datetime ;";


    $result = $pdo->query($sql);
    //if ($result->rowCount() > 0) {
    foreach ($result as $row) {
        $waterlevel[] = $row['water_level_bg'];
        $date[] = $row['datetime'];
```

```
    }
} catch (PDOException $e) {
    echo $e->getMessage();
}

?>
```

File: scatterplot_main.php

In this file, tables and columns are chosen to create scatter plots of one variable against another:

```
<?php include "includes/init.php"?>
<?php include "includes/navigation2.php"?>

<?php include "includes/header.php" ;




if ($_SERVER['REQUEST_METHOD'] == "POST") {
    if (isset($_POST['table'])) {
        $table = $_POST['table'];
    } else {
        $table = "";
    }
}
if (isset($_POST['column'])) {
    $column = $_POST['column'];
} else {
    $column = "";
}
if ($_SERVER['REQUEST_METHOD'] == "POST") {
    if (isset($_POST['table2'])) {
        $table2 = $_POST['table2'];
```

```
    } else {
        $table2 = "";
    }
}
if (isset($_POST['column2'])) {
    $column2 = $_POST['column2'];
} else {
    $column2 = "";
}

if ($_SERVER['REQUEST_METHOD'] == "POST") {
    try {
        $sql1 = "SELECT {$column}  FROM {$table} ;";

        $result = $pdo->query($sql1);
        //if ($result->rowCount() > 0) {
        foreach ($result as $row) {
            $values1[] = $row[$column];
        }
    } catch (PDOException $e) {
        echo $e->getMessage();
    }
}
if ($_SERVER['REQUEST_METHOD'] == "POST") {
    try {
        $sql2 = "SELECT {$column2}
FROM  {$table2};";
        $result = $pdo->query($sql2);
        //if ($result->rowCount() > 0) {
        foreach ($result as $row) {
```

```
            $values2[] = $row[$column2];
        }
    } catch (PDOException $e) {
        echo $e->getMessage();
    }
}
?>
```

The select element uses the system view information_schema to get the fields (column values) for the graph:

```
<select id="table2" class=" left" name="table2" class='form-control'>
                            <option value="" disabled selected>--click
here to choose table--</option>
                            <?php

                            try {
                                $result = $pdo->query("SELECT distinct
table_name
                                FROM information_schema.columns
                                WHERE table_schema = 'public'

                                and table_name in ('well_parent',
'wq', 'construction', 'hydrogeo', 'mw_data', 'trans_aquifers' );");
                                foreach ($result as $row) {
                                    echo "<option
value={$row['table_name']}>{$row['table_name']}</option>";
                                }
                                $table = $row['table_name'];
                            } catch (PDOException $e) {
                                echo "Error: " . $e->getMessage();
                            }
                            ?>
                        </select>
```

Values from the database are passed into javascript arrays via the json_encode function:

```
const variable1 = <?php echo json_encode($values1) ?>;
const variable2 = <?php echo json_encode($values2) ?>;
```

Embedded PHP file: radar2.php

We see the same pattern in this graph page as well:

```
<?php include "includes/init.php"?>
<?php include "includes/navigation2.php"?>

<?php include "includes/header.php" ;




if ($_SERVER['REQUEST_METHOD'] == "POST") {
    if (isset($_POST['sample_no'])) {
        $sample_no = $_POST['sample_no'];
    } else {
        $sample_no = "";
    }
    try {
        $result = $pdo->query("Select ca_meql, mg_meql, na_meql,
k_meql, cl_meql, hco3_alk_meql, so4_meql from wq
        where sample_no = '{$sample_no}';");
           $vararray =$result->fetch();
       //echo $sample_no;




        }
```

```
     catch (PDOException $e) {
        echo "Error: " . $e->getMessage();
    }
}



if ($_SERVER['REQUEST_METHOD'] == "POST") {
    if (isset($_POST['sample_no_b'])) {
        $sample_no_b = $_POST['sample_no_b'];
    } else {
        $sample_no_b = "";
    }
    try {
        $result = $pdo->query("Select ca_meql, mg_meql, na_meql,
k_meql, cl_meql, hco3_alk_meql, so4_meql from wq
        where sample_no = '{$sample_no_b}';");
           $vararray2 =$result->fetch();
       //echo $sample_no_b;









        }
     catch (PDOException $e) {
        echo "Error: " . $e->getMessage();
    }
}
if (isset($_POST['sample_no'])) {
    $sample_no = $_POST['sample_no'];
    if (isset($_POST['sample_no_b'])) {
        $sample_no_b = $_POST['sample_no_b'];
```

```
        echo "<div> <h3 class='text-center'>Sample no $sample_no
(blue) and Sample no $sample_no_b (red)</h3></div>";
} else {
    $sample_no = "";
    $sample_no_b = "";
}
}


    ?>
<select id="sample_no" class=" left" name="sample_no"  class='form-
control'>
                        <option value="" disabled selected>--click
here to choose sample number--</option>
                        <?php


                        try {
                            $result = $pdo->query("select sample_no
from wq;");
                            foreach ($result as $row) {
                                echo "<option
value={$row['sample_no']}>{$row['sample_no']}</option>";
                            }
                            $sample_no = $row['sample_no'];
                        } catch (PDOException $e) {
                            echo "Error: " . $e->getMessage();
                        }
                        ?>


                    </select>
```

JAVASCRIPT, AJAX and CDN Libraries

There are not many instances of raw JavaScript being used. One example is for form validation in the well_construct.php file, where hole and casing diameters and depths of screens were checked for logical consistency (bottom of screen must be lower than the top and less than total depth. The lines of code for checking were triggered by the *onchange* event of the input boxes.

```
<script type="text/javascript">
        function casing_val() {
            //Set the element variables
            let xElement = document.getElementById("casing_diam_mm");
            let yElement = document.getElementById("hole_diam_mm");
            let x = parseFloat(xElement.value); // parseFloat converts
string to floating number
            let y = parseFloat(yElement.value);

            if (x > y) {
                xElement.value = ""; // use the element variable to
set the value
                alert("casing diameter must be less than hole
diameter. Try again.");
            }
        }



        function firstscreen_val() {
            let st1Element =
document.getElementById("screen1_top_dbg");
            let sb1Element =
document.getElementById("screen1_bot_dbg");
            let zElement = document.getElementById("depth_dbg");
            let st1 = parseFloat(st1Element.value);
```

```
            let sb1 = parseFloat(sb1Element.value);
            let z = parseFloat(zElement.value);
            if (st1 > z) {
                alert("Top of screen must be above the total depth.
Try again.");
                st1Element.value = "";
            }

            if (sb1 < st1) {
                alert("Bottom of screen must be lower than the top.
Try again.");
                sb1Element.value = "";

            }

            if (sb1 > z) {
                alert("bottom of screen must be above the total depth.
Try again.");
                sb1Element.value = "";



            }

        }

        function secondscreen_val() {
            let st2Element =
document.getElementById("screen2_top_dbg");
            let sb2Element =
document.getElementById("screen2_bot_dbg");
            let zElement = document.getElementById("depth_dbg");
            let st2 = parseFloat(st2Element.value);
```

```
            let sb2 = parseFloat(sb2Element.value);
            let z = parseFloat(zElement.value);
            if (st2 > z) {
                alert("Top of screen must be above the total depth.
Try again.");
                st2Element.value = ""
            }
            if (st2 > sb2) {
                sb2Element.value = "";
                alert("Bottom of screen must be lower than the top.
Try again.");
            }

            if (sb2 > z) {
                alert("bottom of screen must be above the total depth.
Try again.");
                st2Element.value = ""
            }

        }


        function thirdscreen_val() {
            let st3Element =
document.getElementById("screen3_top_dbg");
            let sb3Element =
document.getElementById("screen3_bot_dbg");
            let zElement = document.getElementById("depth_dbg");
            let st3 = parseFloat(st3Element.value);
            let sb3 = parseFloat(sb3Element.value);
            let z = parseFloat(zElement.value);
            if (st3 > z) {
```

```
                alert("Top of screen must be above the total depth.
Try again.");
            }

            if (sb3 > z) {
                alert("Top of screen must be above the total depth.
Try again.");
            }
            if (st3 > sb3) {
                sb3Element.value = "";
                alert("Bottom of screen must be lower than the top.
Try again.");
            }

        }
    </script>
```

In the mapping module, a great deal of JavaScript was used, and kept in separate files in a JavaScript directory, in the **main.js** file, the contents of which are reproduced here:

```
var initmap;
var lyrOSM;

var lyrTopo = L.tileLayer("https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png");
var lyrImagery = L.tileLayer(
  "https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}"
);

var ctlAttribute;
var ctlScale;

var ctlMeasure;
var ctlEasybutton;
```

```
var ctlSidebar;
var ctlLayers;

var objBasemaps;
var objOverlays;

var mystyle = {
  color: "#0a0100",
  weight: 1,
  fillOpacity: 0,
  fillcolor: "",
};

var mystyle2 = {
  color: "#f51707",
  weight: 1,
  opacity: 1,
};
var recharge_styl = {
  color: "#c9f507",
  weight: 0.5,
  opacity: 0.7,
};
var trb_styl = {
  weight: 1,
  fillOpacity: 0,
};

var initmap = L.map(mapdiv);
```

```
initmap.setView([-0.791, 42.1113], 6);
lyrOSM = L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", {
  attribution:
    '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
}).addTo(initmap);

objBasemaps = {
  "Open Street Maps": lyrOSM,
  "Topo Map": lyrTopo,
  Imagery: lyrImagery,
};

initmap.createPane("left");
initmap.createPane("right");

var recharge = new L.GeoJSON.AJAX("../data/rch_update.geojson", {
  style: recharge_styl,
});

var hydrogeo = new L.GeoJSON.AJAX("../data/merged_Hydrogeo.geojson", {
  style: mystyle2,
});

var watershed = new L.GeoJSON.AJAX("../data/watersheds_transboundary.geojson", {
  style: trb_styl,
});

var transboundary = new L.GeoJSON.AJAX("../data/trans_aquifers.geojson", {
  style: mystyle,
```

```
}).addTo(initmap);

//gw_prod and gw_store_mm
var gw_prod = new L.GeoJSON.AJAX("../data/gw_prod.geojson", {
  style: mystyle,
});

//gw_store_mm
var gw_store_mm = new L.GeoJSON.AJAX("../data/gw_store_mm.geojson", {
  style: mystyle,
});

// transboundary aquifers popup
transboundary.on("data:loaded", function () {
  transboundary
    .bindPopup(function (layer) {
      return `<b>Name: </b> ${layer.feature.properties.name
        .replace("_", " ")
        .replace(/\b\w/g, (s) => s.toUpperCase())} <br>
          <b>Area: </b> ${layer.feature.properties.area_km} km<sup>2</sup> <br>
          <b>Countries: </b> ${layer.feature.properties.countries} <br>
          <b>Population: </b> ${layer.feature.properties.population} <br>
          <b>Annual Raninfall (mm): </b> ${
            layer.feature.properties.annual_rainfall_mm
          } <br>
          `;
    })
    .addTo(initmap);
  initmap.fitBounds(transboundary.getBounds());
```

```
});

// add tooltip for the gw_prod layer
gw_prod.on("data:loaded", function () {
  gw_prod.bindTooltip(function (layer) {
    return `${layer.feature.properties["m3/hr"]} <br>`;
  });
  //   gw_prod.addTo(initmap);
});

// add tooltip for the gw_store_mm
gw_store_mm.on("data:loaded", function () {
  gw_store_mm.bindTooltip(function (layer) {
    return `${layer.feature.properties["storage(mm_depth)"]} <br>`;
  });
  //   gw_store_mm.addTo(initmap);
});

//=============================================================================================
====================
//Basic functions and map controls
//=============================================================================================
====================

// mouse coordinates
initmap.on("mousemove", function (e) {
  var str =
    "Latitude: " +
    e.latlng.lat.toFixed(4) +
```

```
    "   Longitude: " +
    e.latlng.lng.toFixed(4) +
    "   Zoom Level: " +
    initmap.getZoom();

  $("#map_coords").html(str);
});

objOverlays = {
  "Transboundary aquifers": transboundary,
  "BGS Hydrogeology": hydrogeo,
  "Groundwater Productivity": gw_prod,
  "Groundwater storage (mm)": gw_store_mm,
  Watersheds: watershed,
  Recharge: recharge,
};

//   layer controller
ctlLayers = L.control.layers(objBasemaps, objOverlays).addTo(initmap);

// scale control
ctlScale = L.control
  .scale({ position: "bottomleft", imperial: false, maxWidth: 200 })
  .addTo(initmap);

//================================================================================
====================
// left sidebar
```

```
//=============================================================================================
====================

const left = '<h3 class="header">Zoom to layer</h3>';

const aquifer = [

  "Rift Aquifer",

  "Mt Elgon Aquifer",

  "Merti Aquifer",

  "Karoo Sedimentary Aquifer",

  "Dawa Aquifer",

  "Jubba Aquifer",

  "Shabelle Aquifer",

  "kilimanjaro Aquifer",

];

const aquiferId = aquifer.map((a) => a.replace(/ /g, "_").toLowerCase());

let contents = `

          <div class="zoom-to-aquifer">

             ${aquifer

               .map(

                 (a) =>

                   `<button id="${a

                     .replace(/ /g, "_")

                     .toLowerCase()}" class="form-control btn btn-primary btn-block btnspace">${a}</button>`

               )

               .join("")}

          </div>`;


//   left control

L.control

  .slideMenu(left + contents, {
```

```
    position: "topleft",
    menuposition: "topleft",
    width: "30%",
    height: "100%",
    direction: "horizontal",
    changeperc: 50,
    delay: 0.5,
    icon: "fas fa-bars",
    hidden: false,
  })
  .addTo(initmap);

//  leaflet sidebar show button
$(".leaflet-bar-part").on("click", () => {
  $(".leaflet-top.leaflet-left").css("left", "30%");
  $(".leaflet-bottom.leaflet-left").css("left", "30%");
  $(".leaflet-bar-part").hide();
});

// leaflet sidebar close button
$(".leaflet-menu-close-button").on("click", () => {
  $(".leaflet-bar-part").show();
  $(".leaflet-bottom.leaflet-left").css("left", "0%");
  $(".leaflet-top.leaflet-left").css("left", "0%");
});

// map aquiferid to zoom to the aquifer using above logic
aquiferId.forEach((id) => {
  $(`#${id}`).on("click", () => {
```

```
    const requiredLayer = transboundary.getLayers().filter((layer) => {
      return layer.feature.properties.name === id;
    });
    //   console.log({ requiredLayer, id });
    initmap.fitBounds(requiredLayer[0].getBounds());
  });
});

//   distance and area measurement
ctlMeasure = L.control.measure({
  position: "topright",
  primaryLengthUnit: "meters",
  primaryAreaUnit: "sqmeters",
});
ctlMeasure.addTo(initmap);

//=============================================================================================
// leaflet sideby side map initialize
//=============================================================================================

// layer names
var layerNames = [
  "Transboundary aquifers",
  "BGS Hydrogeology",
  "Watersheds",
  "Recharge",
  "Groundwater Productivity",
```

```
  "Groundwater storage (mm)",
];

// create a button in top left corner using font awesome icon
var compareBtn = L.control({
  position: "topleft",
});
compareBtn.onAdd = function (map) {
  this._div = L.DomUtil.create("div", "leaflet-bar leaflet-control");
  var btn = `
      <div data-toggle='tooltip' data-placement='right' title='Compare Maps' class='leaflet-bar-part leaflet-bar-part-singlmap-btn'>
        <a class='leaflet-bar-part leaflet-bar-part-singlmap-btn' data-toggle="modal" data-target="#compareMap">
          <i class="fa-solid fa-sliders"></i>
        </a>
      </div>`;

  this._div.innerHTML = btn;
  return this._div;
};
compareBtn.addTo(initmap);

// add options to the first selector
const firstSelector = document.getElementById("firstSelector");
const secondSelector = document.getElementById("secondSelector");

layerNames.forEach((layer) => {
  const option = document.createElement("option");
```

```
  option.value = layer;
  option.text = layer;
  firstSelector.appendChild(option);
});

layerNames.forEach((layer) => {
  const option = document.createElement("option");
  option.value = layer;
  option.text = layer;
  secondSelector.appendChild(option);
});

// get the selected layer name
const getSelectedLayer = (selector) => {
  return selector.options[selector.selectedIndex].value;
};

// compare button click event
var sidebyside;
$(".compare-map-btn").on("click", () => {
  const firstLayer = getSelectedLayer(firstSelector);
  const secondLayer = getSelectedLayer(secondSelector);
  initmap.invalidateSize();

  console.log({ firstLayer, secondLayer });

  if (sidebyside) initmap.removeControl(sidebyside);

  // get the selected layers
```

```
  const firstLayerData = objOverlays[firstLayer];
  const secondLayerData = objOverlays[secondLayer];

  // remove all the layers
  initmap.eachLayer((layer) => {
    if (layer instanceof L.GeoJSON) {
      console.log("layer", layer);
      initmap.removeLayer(layer);
    }
  });

  firstLayerData.addTo(initmap);

  // set the pane for the second layer
  secondLayerData.options.pane = "right";
  secondLayerData.eachLayer(function (layer) {
    layer.options.pane = "right";
    layer.defaultOptions.pane = "right";
  });
  secondLayerData.addTo(initmap);

  // create a side by side map
  sidebyside = L.control.sideBySide(firstLayerData, secondLayerData);
  sidebyside.addTo(initmap);
});

// remove compare button click event
$(".compare-map-cancel-btn").on("click", () => {
  initmap.eachLayer((layer) => {
```

```
    if (layer instanceof L.GeoJSON) {
      initmap.removeLayer(layer);
    }
  });

  transboundary.addTo(initmap);

  // remove the side by side map
  initmap.removeControl(sidebyside);
});

//=============================================================================================
// add legend to the map
//=============================================================================================

// add legend to the map
var legend = L.control({ position: "bottomright" });
legend.onAdd = function (map) {
  var div = L.DomUtil.create("div", "info legend");
  div.innerHTML += "<h4>Legend</h4>";
  div.innerHTML +=
    '<i style="background: #000000"></i>Transboundary Aquifers<br>';
  div.innerHTML += '<i style="background: #ff0000"></i>Hydrogeology<br>';
  div.innerHTML += '<i style="background: #c9f507"></i>Recharge<br>';
  div.innerHTML += '<i style="background: #0000ff"></i>Watersheds<br>';
  return div;
};
```

```
legend.addTo(initmap);
```

The application depends on several libraries that are supplied by content delivery networks (CDNs) whose links must be included in the files they are used in. Below is the list of these links and their URLs. Note that in many cases, both a CSS file and a JS file are associated with the libraries

BOOTSTRAP (CSS style library)

```
<link
    rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"
    integrity="sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N"
    crossorigin="anonymous"
  />
<script
    src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-Fy6S3B9q64WdZWQUiU+q4/2Lc9npb8tCaSX9FK7E8HnRr0Jz8D6OP9dO5Vg3Q9ct"
    crossorigin="anonymous"
  ></script>
  <link
    rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"
    integrity="sha384-xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N"
    crossorigin="anonymous"
  />
```

JQUERY (JavaScript library for AJAX and browser element manipulation)

```
<script
    src="https://cdn.jsdelivr.net/npm/jquery@3.5.1/dist/jquery.slim.min.js"
    integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj"
    crossorigin="anonymous"
  ></script>
```

LEAFLET (mapping library)

```
<link
    rel="stylesheet"
    href="https://unpkg.com/leaflet@1.7.0/dist/leaflet.css"
  />
<script src="https://unpkg.com/leaflet@1.7.0/dist/leaflet.js"></script>
  <link
    rel="stylesheet"
    href="./JavaScript/plugins/L.Control.SlideMenu.css"
  />
```

CHART JS (graphing library)

```
<script type="text/javascript"
src="https://cdn.jsdelivr.net/npm/chart.js/dist/chart.umd.min.js"></script>
  <link rel="stylesheet" href="css/style.css">
```

Some of the plugins for these libraries, which offer increased functionality, are not supplied with connections to a CDN, but rather downloaded and kept in the Javascript folder of each respective module. This was done because the reliability of the repositiories is unknown, and might be dropped in the future. Ecamples are:

```
<!-- leaflet measure  -->
  <script src="./JavaScript/plugins/leaflet-measure.js"></script>
```

```
    <!-- leaflet heatmap  -->
    <script src="./JavaScript/plugins/leaflet-heat.js"></script>
<script src="./JavaScript/plugins/L.Control.SlideMenu.js"></script>
    <script src="./JavaScript/plugins/leaflet-side-by-side-v2.js"></script>
    <script src="src/plugins/Leaflet.PolylineMeasure.js"></script>
    <script src="src/plugins/easy-button.js"></script>
    <script src="src/plugins/leaflet-providers.js"></script> -->
    <script src="./JavaScript/plugins/leaflet.ajax.min.js"></script>
```

Chart JS – typical structure:

```
// setup

        const waterlevel = <?php echo json_encode($waterlevel) ?>;
        const date = <?php echo json_encode($date) ?>;

        const data = {
            labels: date,
            datasets: [{
                label: 'water level (m) below ground',
                data: waterlevel,
                backgroundColor: [
                    //'rgba(255, 26, 104, 0.2)',
                    'rgba(54, 162, 235, 0.2)',
                    //'rgba(255, 206, 86, 0.2)',
                    //'rgba(75, 192, 192, 0.2)',
                    //'rgba(153, 102, 255, 0.2)',
                    //'rgba(255, 159, 64, 0.2)',
                    //'rgba(0, 0, 0, 0.2)'
                ],
```

```
        borderColor: [
            //'rgba(255, 26, 104, 1)',
            'rgba(54, 162, 235, 1)',
            //'rgba(255, 206, 86, 1)',
            //'rgba(75, 192, 192, 1)',
            //'rgba(153, 102, 255, 1)',
            //'rgba(255, 159, 64, 1)',
            //'rgba(0, 0, 0, 1)'
        ],
        borderWidth: 1,
        tension: 0.4
    }]




};



// config
const config = {
    type: 'line',
    data,
    options: {

        scales: {
            y: {
                beginAtZero: false,
```

```
        title: {
            display: true,
            text: 'depth below ground'
        }
    }

},
plugins: {

    title: {
        display: true,
        text: 'Well no <?php echo $mw_id ?>'
    },

    zoom: {
        zoom: {
            wheel: {
                enabled: true
            }

        },
        pan: {
            // pan options and/or events
            enabled: true,
            mode: 'xy'
            //limits: {
            // axis limits
        }
```

```
            }
          }


      }
    }


    // render init block
    const myChart = new Chart(
      document.getElementById('myChart'),
      config
    );

    // Instantly assign Chart.js version
    const chartVersion = document.getElementById('chartVersion');
    chartVersion.innerText = Chart.version;
    Chart.register(zoomPlugin);
</script>
```