Note: For the following queries, assume that **$variable** is a placeholder for the corresponding variable described by the name. Also assume that **variable_id.nextval** refers to a sequence generated & used.

Note: For each functionality/feature, consider the listed queries to be done in that order.

Login Module

The login module was used by having the user input an username and password into the text field of the loginModule.html page. Then in the LoginModule.php file we check to see if the fields are filled or not then we check to see if the username and password they entered exists in the database through our sql querie statments. Then we start a session to save the information of the currently logged in user.

After that we are redirected to our main page. In our php file for main page, we first start a session and check to see that the user is the same as the one that had just logged in. Then we also get the status of the logged in user and depending on whether they are an admin, scientist, or data curator they will be displayed different options of what modules they can access.

Every type of user is allowed to logout and change their passwords & update their personal information.

The main sql queries used are:

1) **To see if the username and password inputted is valid**

SELECT * FROM users WHERE users.user_name = '$_POST[username]' and users.password = '$_POST[password]'

Update Info & PW module

This module allows users to change their passwords and update their associated first/last name, address, email & phone number. To change their password, the user has to enter their old password then their new one and confirm once again on their new password. The php file first checks to see if the user currently trying to access the change pw page is a user that has been logged in, this is done by sessions. Then it will check the fields of the passwords, to see if it is empty or if they new password has been incorrectly inputted. Once that is done we use our sql statement to change the user's password. To update personal information, it is done the same way as updating a person in the user module (explained below).

The main sql queries used are:

1) **To update the password**

UPDATE users SET password='$password1' WHERE user_name='$username' AND person_id=$personid

2) **To update personal information**

See "update a person" in the user module below.

User Module

This module allows admins to view, add update & remove persons & users.
The user module used 7 main functions:

1) **validatePersonID** – Checks if the person id provided is an existing person
    a. "SELECT * FROM persons WHERE person_id= $person_id"
2) **validateEmail** – Checks if the email provided is already used in the database

    a. “SELECT * FROM persons WHERE email = $email”
3) **removeSubscriptions –** Removes all subscriptions for a given person id
    a. “SELECT * FROM subscriptions WHERE person_id= $person_id”
    b. “DELETE FROM subscriptions WHERE person_id = $person_id”
4) **removeUser –** Removes a given user from the database from the user name provided
    a. “DELETE FROM users WHERE user_name = $username”
5) **removeUserFromPersonID –** Removes all user accounts associated with the provided person id
    a. “DELETE FROM users WHERE person_id = $person_id”
6) **removePerson –** Removes the person with the given person id from the database
    a. “DELETE FROM persons WHERE person_id = $person_id”
7) **validateUsername –** Checks if the given username is an existing user in the database
    a. “SELECT * FROM users WHERE user_name = $username”

To add a new person:

a. **validateEmail** was used to ensure the unique constraint for email was not violated. If it already exists, do not continue.
b. “INSERT INTO persons VALUES (person_id.nextval, $first_name, $last_name, $address, $email, $phone_number)”

To update a person:

a. **validatePersonID** was used to ensure the an existing person id was entered before updating a person. If not, do not continue.
b. **validateEmail** was used to ensure the unique constraint for email was not violated. If it already exists, do not continue.
c. “UPDATE persons SET first_name = $first_name, last_name = $last_name, address = $address, email = $email, phone = $phone WHERE person_id = $person_id”
    a. Only the fields that were entered in the form are included in the query

To remove a person:

a. **validatePersonID** was used to ensure the an existing person id was entered before removing a person. If not, do not continue.
b. **removeSubscriptions** was used to remove any subscriptions the person had before removing the person ensure the integrity constraint was not violated
c. **removeUserFromPersonID** was used to remove any user accounts associated with the person to ensure the integrity constraint was not violated
d. **removePerson** was used to remove the person

To display the contents of the persons table:

a. The function **get_all_persons** was used to first retrieve all the records in the person table – “SELECT * FROM persons”

To add a new user account:

a. **validateUsername** was used to ensure the unique constraint was not violated. If the username already exists, do not continue.
b. **validatePersonID** was used to ensure the an existing person id was entered. If person id entered doesn't exist, do not continue.
c. “INSERT INTO users VALUES ($username, $password, $role, $person_id, to_date($date_registered, 'dd-mm-yyyy hh24:mi:ss'))”

To update a user account: Allows admin to update everythig except the person_id associated with the user account

a. **validateUsername** was used to ensure an existing username was entered before updating. If not, do not continue.
b. "UPDATE users SET password = $password, role = $role WHERE user_name = $username"
    a. Only the fields that were entered in the form are included in the query

To remove a user account:

a. **validateUsername** was used to ensure an existing username was entered before removing. If not, do not continue.
b. **removeUser** was used to remove the user account

To display the contents of the users table:

a. The function **get_all_users** was used to first retrieve all the records in the users table – "SELECT user_name, password, role, person_id, to_char(date_registered, 'dd/mm/YYYY hh24:mi:ss') as date_registered FROM users"

Sensor Module

This module allows admins to view, add & remove sensors.
The sensor module used 4 main functions:

1) **validateSensor –** Checks if the provided sensor id already exists in the database
    a. "SELECT * FROM sensors WHERE sensor_id = $sensor_id"
2) **removeSubscriptionsFromSensorID –** Removes all subscriptions associated with the given sensor id
    a. "DELETE FROM subscriptions WHERE sensor_id = $sensor_id"
3) **removeSensor –** Removes the sensor with the given sensor id from the database
    a. "DELETE FROM sensors WHERE sensor_id = $sensor_id"
4) **removeData** – Removes all images, audio recordings & scalar data that correspond to the sensor id provided
    a. "DELETE FROM images WHERE sensor_id = $sensor_id"
    b. "DELETE FROM audio_recordings WHERE sensor_id = $sensor_id"
    c. "DELETE FROM scalar_data WHERE sensor_id = $sensor_id"

To add a sensor:

a. "INSERT INTO sensors VALUES (sensor_id.nextval, $location, $type, $description)"

To remove a sensor:

a. **validateSensor** was used to ensure an existing sensor id was entered before removing. If not, do not continue.
b. **removeSubscriptionsFromSensorID** was used to remove any subscriptions associated with the sensor to ensure the integrity constraint was not violated
c. **removeData** was used to remove all images, audio recordings & scalar data that correspond the sensor id before removing the sensor to ensure no integrity constraints are violated
d. **removeSensor** was used to remove the sensor

To display the contents of the sensors table:

a. The function **get_all_sensors** was used to first retrieve all the records in the sensors table – “SELECT * FROM sensors”

Upload Module

The upload module allows data curators to upload their data, which are images, audio files and csv files. Images are only accepted in jpg format, audio files are only accepted as wav files and csv files must be in format of sensor id, date, value.

Images:

When uploading images, the user is given fields in which they must fill. The first is what image they want to upload and this is done through the html input file type form. Next is the sensor id of the said image they are going to upload. After that is the date, then the description.

Once that is done it will go to our Image.php file which will first use sessions to check that the user currently logged in is a data curator. Then it will get the information of the image that the user wants to upload. It will then check to see if it exceeds 64kb and return an error if it does, then it will check to see if it is in proper format (jpg) and returns an error if it isn't. It will also check to see if a valid sensor id has been inputted.

This is done through the use of the file 'size' and file 'type'. Once it passes the checks, it will get ready to upload the information into the images table. First it will get a new unique id from images by using an sql statement which gets the max image id from images then it will continue on from that. Next it will start inputting the blob file into recorded data and thumbnail respectively. We upload this information by first have empty blobs in the sql statement, then we fill it up by using oci_new_descriptor, oci_bind_by_name and savefile.

The main sql queries used are:

1) **To get unique image_id**

   SELECT MAX(image_id) FROM images

2) **To insert into images table**

   INSERT INTO images(image_id, sensor_id, date_created, description, thumbnail, recoreded_data) VALUES ('$image_id', '$sensor_id', to_date('$date', 'dd/mm/YYYY hh24:mi:ss'), '$description', empty_blob(), empty_blob()) returning thumbnail, recoreded_data into :thumbnail, :recoreded_data

Audio:

The audio php uploading file is similar to that of the image one. The user first fills in the fields of which are what audio file is to be uploaded, the sensor id, the date, length, and description. Then after the fields are filled, it will go to the audio.php file and run the code which will first check again through sessions to see if the currently logged in user is a valid user or not.

Then it will see if an audio file has been uploaded or not. The next check after that is to see if it is a wav file or not. It will also check to see if a valid sensor id has been inputted. If it does not pass these checks, it will prompt the user to retry. Next it will get ready to upload the information into the database. This is done similarily to how we upload images, with first getting an unique audio id, then using an empty blob and oci_new_descriptor, oci_bind_by_name and savefile to upload the audio file.

The main sql queries used are:

1) **To get an unique recording_id**

SELECT MAX(recording_id) FROM audio_recordings

2) **To insert into audio_recordings**

INSERT INTO audio_recordings(recording_id, sensor_id, date_created, length, description, recorded_data) VALUES ('$recording_id', '$sensor_id', to_date('$date', 'dd/mm/YYYY hh24:mi:ss'), '$length', '$description', empty_blob()) returning recorded_data into :recorded_data

CSV Files:

What we upload to the database is a csv file of format sensor id, date, and value. These are all on one line separated by a newline and we can have multiple entries by seperating it with newlines. After a csv file is uploaded, in our php file we first check to see if the currently logged in user is a valid data curator. Then we check to see if the file uploaded is valid, i.e. has a size greater than 0 bytes.

Once the checks pass, we then will use an array to store our sql statments that we will execute in a loop later in the code. First we will read from the csv file using fopen and fgetcsv which will then put the sensor id, date and value into a variable. We will then parse these variables into sql statements and then will put it into our array of sql queries. After reading the file is all done, we will then have a loop which will execute all the statements that have been stored in our array.

The main sql queries used are:

1) **To insert the scalar data from csv file**

INSERT INTO scalar_data VALUES ('$new_id', '$sensor_id', to_date('$date', 'dd/mm/yyyy hh24:mi:ss'), '$value')

Search Module

This module allows scientists to search for images, audio recordings & scalar data that are created within a time period (mandatory) and also by keywords, sensor type, or sensor location (all of which are optional) that correspond to sensors that they have subscribed to. It returns any records whose description or whose corresponding sensor's description contains ANY of the keywords AND satisfies the sensor type & sensor location if provided. This module displays thumbnails of the images in the results & it allows scientists to download full sized images, the audio recording as well as a csv file of scalar data.

To search:
We did 3 separate queries: one for each of images, audio recordings & scalar data:
We basically concatenated every search option with AND and to search for matching keywords, we search for sensor description field & image/audio recording description fields & used OR to concatenate the keyword queries.

- For Images:
  “SELECT i.image_id, i.sensor_id, to_char(i.date_created, 'dd/mm/YYYY hh24:mi:ss') as date_created, i.description, i.thumbnail, i.recoreded_data FROM images i, sensors s, subscriptions t WHERE t.person_id = $person_id AND s.sensor_id = t.sensor_id AND i.sensor_id = s.sensor_id AND s.sensor_type = $sensor_type AND s.location = $sensor_location AND (lower(i.description) LIKE '%$keyword1%' OR lower(s.description) LIKE '%$keyword1%' OR lower(i.description) LIKE '%$keyword2%' OR lower(s.description) LIKE '%$keyword2%') AND i.date_created >= to_date($start_date,

'dd/mm/YYYY hh24:mi:ss') AND i.date_created <= to_date($end_date, 'dd/mm/YYYY hh24:mi:ss')"

- For audio recordings:
  "SELECT a.recording_id, a.sensor_id, to_char(a.date_created, 'dd/mm/YYYY hh24:mi:ss') as date_created, a.length, a.description, a.recorded_data FROM audio_recordings a, sensors s, subscriptions t WHERE t.person_id = $person_id AND s.sensor_id = t.sensor_id AND a.sensor_id = s.sensor_id AND s.sensor_type = $sensorType AND s.location = $sensor_location AND (lower(a.description) LIKE '%$keyword1%' OR lower(s.description) LIKE '%$keyword1%' OR lower(a.description) LIKE '%$keyword2%' OR lower(s.description) LIKE '%$keyword2%') AND a.date_created >= to_date($start_date, 'dd/mm/YYYY hh24:mi:ss') AND a.date_created <= to_date($end_date, 'dd/mm/YYYY hh24:mi:ss')"

- For scalar data:
  "SELECT c.id, c.sensor_id, to_char(c.date_created, 'dd/mm/YYYY hh24:mi:ss') as date_created, c.value FROM scalar_data c, sensors s, subscriptions t WHERE t.person_id = $person_id AND s.sensor_id = t.sensor_id AND c.sensor_id = s.sensor_id AND s.sensor_type = $sensor_type AND s.location = $sensor_location AND (lower(s.description) LIKE '%$keyword1%' OR lower(s.description) LIKE '%$keyword2%') AND c.date_created >= to_date($start_date, 'dd/mm/YYYY hh24:mi:ss') AND c.date_created <= to_date($end_date, 'dd/mm/YYYY hh24:mi:ss')"

  - $start_date & $end_date are assumed to be in the form dd/mm/YYYY hh24:mi:ss, or example '31/08/2016 22:34:54'

  - "lower(i.description) LIKE $keyword1 OR lower(s.description) LIKE $keyword1" is done for every keyword and connected via an "OR"

Subscribe Module
This module allows scientists to view all available sensors and subscribe/unsubscribe from sensors by checking/unchecking and pressing submit.

The table is first generated by calling *get_sensor()* and putting the values of each sensor (id and location) into the table.
Each sensor is also checked for whether the current user is subscribed to it or not by calling *is_subscribed()* on each sensor, and checking the ones that are subscribed and keeping the unsubscribed ones unchecked.
After the user makes the selection and presses submit, *add_subscription()* and *remove_subscription()* functions are called on each sensor to update the user's subscription.

1. Subscribing to a sensor
   This is done by the *add_subscription()* function and is called whenever a checked row is encountered in the table after a POST.

   The function first checks if the person is already subscribed to that sensor with this query:

*SELECT * FROM subscriptions*
WHERE sensor_id = $sensor_id AND person_id = $person_id

If there are already subscribed, the function will exit. If not, the function will add the sensor to that persons' subscription list with this query:

INSERT INTO subscriptions (sensor_id, person_id)
VALUES ($sensor_id,$person_id)

2. Unsubscribing from a sensor
   This is done by the *remove_subscription()* function in a very similar way to the add_subscription above.
   But instead it checks if the person is already unsubscribed and removes the subscription if the person is still subscribed.

   *SELECT * FROM subscriptions*
   WHERE sensor_id = $sensor_id AND person_id = $person_id

   DELETE FROM subscriptions
   WHERE sensor_id = $sensor_id AND person_id = $person_id

3. get_sensors()
   This function is used to populate the table with all available sensors.
   *SELECT * FROM sensors*

4. get_subscribed_sensors()
   This function is used to get all sensors the user is already subscribed to.
   *SELECT * FROM subscriptions*
   WHERE person_id=$person_id

5. is_subscribed()
   Similar to the above function, however this just checks if the user is subscribed to a single sensor.
   *SELECT * FROM subscriptions*
   WHERE sensor_id=$sensor_id AND person_id=$person_id

Data analysis Module
This module allows the user to select a year, time window and one of the currently subscribed sensor as the input. And pressing submit will generate the report corresponding to those input parameters.
After the report is initially generated, the user can choose to “drill down” by pressing the button that corresponds to the area they want to drill down to.

There is no actual OLAP implementation done, however the module still allows the user to do similar drill down/roll up operations in a similar manner.

Since the user only selects from subscribed sensors, the get_subscribed_sensors() function from above is used to grab the list of sensors. If none are available it will display as such in the dropdown menu.

After the user inputs all the parameters and presses the submit button, the inputs from the POST is processed and the function corresponding to the input is called. (All which has very similar sql queries with each other so not all of them will be listed here.)

For example *data_analysis_month()* function may be called whenever the user selects the month option as the time window.

The function simply does the following sql query to get all the required data.
Which aggregates the Max,Min,Avg from whatever scalar data that matches the conditions.

```
SELECT MAX(value),MIN(value),AVG(value), EXTRACT(month FROM date_created) FROM scalar_data
WHERE $sensor_id=sensor_id AND
$year=EXTRACT(year FROM date_created)
GROUP BY EXTRACT(month FROM date_created)
ORDER BY EXTRACT(month FROM date_created)
```

If the function was called from a roll up operation, The module will call the function that is 1 level higher on the time window hierarchy. (e.g. *data_analysis_quarter()* will be called)

If the same function was called from doing a drill down operation (e.g. clicking the button from the quarter table) the optional *$quarter* parameter will be passed on as well. The resulting sql query will also have an extra condition to reflect that as well.

```
SELECT MAX(value),MIN(value),AVG(value), EXTRACT(month FROM date_created) FROM scalar_data
WHERE $sensor_id=sensor_id AND
$year=EXTRACT(year FROM date_created) AND
$quarter=to_number(to_char(date_created,'Q'))
GROUP BY EXTRACT(month FROM date_created)
ORDER BY EXTRACT(month FROM date_created)
```

After the sql queries are made, all the required data is simply placed into the table.