CPU/IO USAGE SUMMARY PROJECT REPORT

SPONSOR: SHANE GRIFFITH/ACXIOM

301 E DAVE WARD DR.

CONWAY, AR 72032

UNIVERSITY OF ARKANSAS AT LITTLE ROCK

CAPSTONE 2

INSTRUCTORS: ELIZABETH PIERCE AND BRUCE BAUER

ALEXANDER SOHN

5/4/2022

TABLE OF CONTENTS

Overview

The CPU/IO Usage Summary Project is a simple yet complicated project. To explain, our team at Acxiom has a client that sends us data in which we fix/complete and overall make better for their business purposes. The "solution" that does this runs 24/7 and is constantly updating tables and sending out files for the client to utilize. The client has teams/groups across the world that run campaigns and reports to gather information from the database to see the success of marketing campaigns, individual store locations, and future business decisions. The whole goal for the project is to provide our Production Support team with the proper tool to look and compare CPU and IO usage between the different groups that use our server, which houses our database. This will include a months' worth of data for our team to look at by day. The importance of this capture time is if we miss our SLA (must finish certain processing by certain time), then we can go back and see if a certain group had over average server usage (that caused our processing to be delayed and non-efficient).

Problem

The current problem with how we capture this data is it is manually done through a SQL query that takes 1 to 2 minutes to process, which is then needed to be moved to excel and made into a usable graph. This only allows for one person to view it and does not provide historical data.

Solution

In all, our solution consists of a webpage that contains two graphs, one for CPU usage and one for IO usage, and a date selection of some sort (drop down or list) that will update the graph. This webpage will be housed on our current inhouse built website, that contains other tools we use daily to help with production support.

Past State

Our solution to this problem is as follows and shown in the current state section. We have built a new SQL query using the old one and techniques from an old graph tool to make the data we capture compatible for our website that we are hosting the data on. First, we put the groups that utilize our server into groups to consolidate them by their type (Direct Access, Business Objects, Acxiom, and Production). Then once the query was built and working, we put it in a create table statement and into a .sql file. Above the create table, we have a delete table statement(important later on).

SQL Code to build table of one months' worth of CPU Data.

```
DROP TABLE IF EXISTS asohn.cpu_usage_summary_tb;

create table asohn.cpu_usage_summary_tb as
select start_hr, sum(MCY_MCCS_BO) MCY_MCCS_BO, sum(BLM_BCM_BO) BLM_BCM_BO, sum(MCY_MCCS_DA) MCY_MCCS_DA, sum(BLM_BCM_DA) BLM_BCM_DA, sum(PRODUCTION) PRODUCTION, sum(ACXIOM) ACXIOM
FROM
(
SELECT START_HR,
DECODE(sub_category,'MCY/MCCS-BO',sum_cpu_minutes,0) MCY_MCCS_BO,
DECODE(sub_category,'BLM/BCM-BO',sum_cpu_minutes,0) BLM_BCM_BO,
DECODE(sub_category,'MCY/MCCS-DA',sum_cpu_minutes,0) MCY_MCCS_DA,
DECODE(sub_category,'BLM/BCM-DA',sum_cpu_minutes,0) BLM_BCM_DA,
DECODE(sub_category,'PRODUCTION',sum_cpu_minutes,0) PRODUCTION,
DECODE(sub_category,'ACXIOM',sum_cpu_minutes,0) ACXIOM
FROM
(select start_hr, schema_name, ora_user_category, sub_category, sum_cpu_minutes, sum_io_bytes
from
(
select
start_hr,
schema_name,
ora_user_category,
case
  when ora_user_category = 'BUS OBJECTS' and schema_name in ('BCM_BO','BLM_BO') then 'BLM/BCM-BO'
  when ora_user_category = 'BUS OBJECTS' and schema_name in ('FDS_BO','MCE_BO') then 'MCY/MCCS-BO'
  when ora_user_category = 'DA_USER' and company in ('BCOM','BLM') then 'BLM/BCM-DA'
  when ora_user_category = 'DA_USER' and nvl(company,'UNK') not in ('BCOM','BLM') then 'MCY/MCCS-DA'
  else ora_user_category
end sub_category,
round(sum(cpu_minutes),5) sum_cpu_minutes,
sum(io_bytes)             sum_io_bytes
from
(
SELECT
  a.schema_name, B.ORA_USER_CATEGORY,to_char(CAPTURE_TIME,'YYMMDDHH24') start_hr, x.company,
  nvl(DELTA_CPU_TIME/60000000,0) cpu_minutes,
  nvl((delta_read_io_bytes+delta_write_io_bytes),0) io_bytes
FROM
  fdssys.ACX_ASH_SNAP_TB  a,
  FDSGLOBAL.ORA_USER_CATEGORY_TB b,
  FDSRPTS.CONTACTS_WWW_TB x
WHERE
  b.ora_user_category <> 'SYSTEM'
  and a.schema_name not in ('BO_AUDITORXI3','DIV_UNICA','UNICA','UNICA2','MCY_BOCMS_A')
  and trunc(a.CAPTURE_TIME) between TO_DATE(SYSDATE-31, 'DD-MON-YY') and TO_DATE(SYSDATE-1, 'DD-MON-YY')
  and a.SCHEMA_NAME = b.username
  and a.schema_name = upper(x.user_id (+))
)
group by
start_hr,
schema_name,
ora_user_category,
case
  when ora_user_category = 'BUS OBJECTS' and schema_name in ('BCM_BO','BLM_BO') then 'BLM/BCM-BO'
  when ora_user_category = 'BUS OBJECTS' and schema_name in ('FDS_BO','MCE_BO') then 'MCY/MCCS-BO'
  when ora_user_category = 'DA_USER' and company in ('BCOM','BLM') then 'BLM/BCM-DA'
  when ora_user_category = 'DA_USER' and nvl(company,'UNK') not in ('BCOM','BLM') then 'MCY/MCCS-DA'
  else ora_user_category
end
)
where 1=1
and sub_category not in ('BUS OBJECTS','TABLEAU','UNICA')
order by start_hr, ora_user_category, sub_category
))
GROUP BY START_HR
ORDER BY START_HR;
```

Past State Cont.

After this SQL was built out, we looked at the existing system of selecting data, creating arrays, building arrays with the data, and then using a query to create labels for the graph and modified to our own. These are the arrays the webpage uses to make the graph with the selected days' worth of data.

PHP and SQL that will build arrays with data

```php
<?php
$database_queries_query = oci_parse($c,"select * from ctoenn.cpu_usage_test_tb
where start_hr between '21110912' and '21111012'");

oci_execute($database_queries_query, OCI_DEFAULT);

//ARRAYS FOR SUBCATEGORIES
$mcy_mccs_bo_array = array();
$blm_bcm_bo_array = array();
$mcy_mccs_da_array = array();
$blm_bcm_da_array = array();
$production_array = array();
$acxiom_array = array();
$array_index = 0;

// build arrays with results
while ($database_queries_result = oci_fetch_array($database_queries_query))
{
    $mcy_mccs_bo_array[$array_index] = $database_queries_result['MCY_MCCS_BO'];
    $blm_bcm_bo_array[$array_index] = $database_queries_result['BLM_BCM_BO'];
    $mcy_mccs_da_array[$array_index] = $database_queries_result['MCY_MCCS_DA'];
    $blm_bcm_da_array[$array_index] = $database_queries_result['BLM_BCM_DA'];
    $production_array[$array_index] = $database_queries_result['PRODUCTION'];
    $acxiom_array[$array_index] = $database_queries_result['ACXIOM'];
  $array_index++;
}

// get the labels for the database queries query
$db_queries_lbl_query = oci_parse($c,"select TO_CHAR(TRUNC(labels,'HH24'),'Mon DD - HH am') HR from
(select to_date(start_hr, 'YYMMDDHH24') labels from ctoenn.cpu_usage_test_tb
where start_hr between '21110912' and '21111012')");

oci_execute($db_queries_lbl_query, OCI_DEFAULT);

$lbl_array = array();
$lbl_index = 0;
while ($db_queries_lbl_result = oci_fetch_array($db_queries_lbl_query))
{
    $lbl_array[$lbl_index] = $db_queries_lbl_result['HR'];
  $lbl_index++;
}
//ADD SECTION FOR IO USAGE
```

Past State Cont.

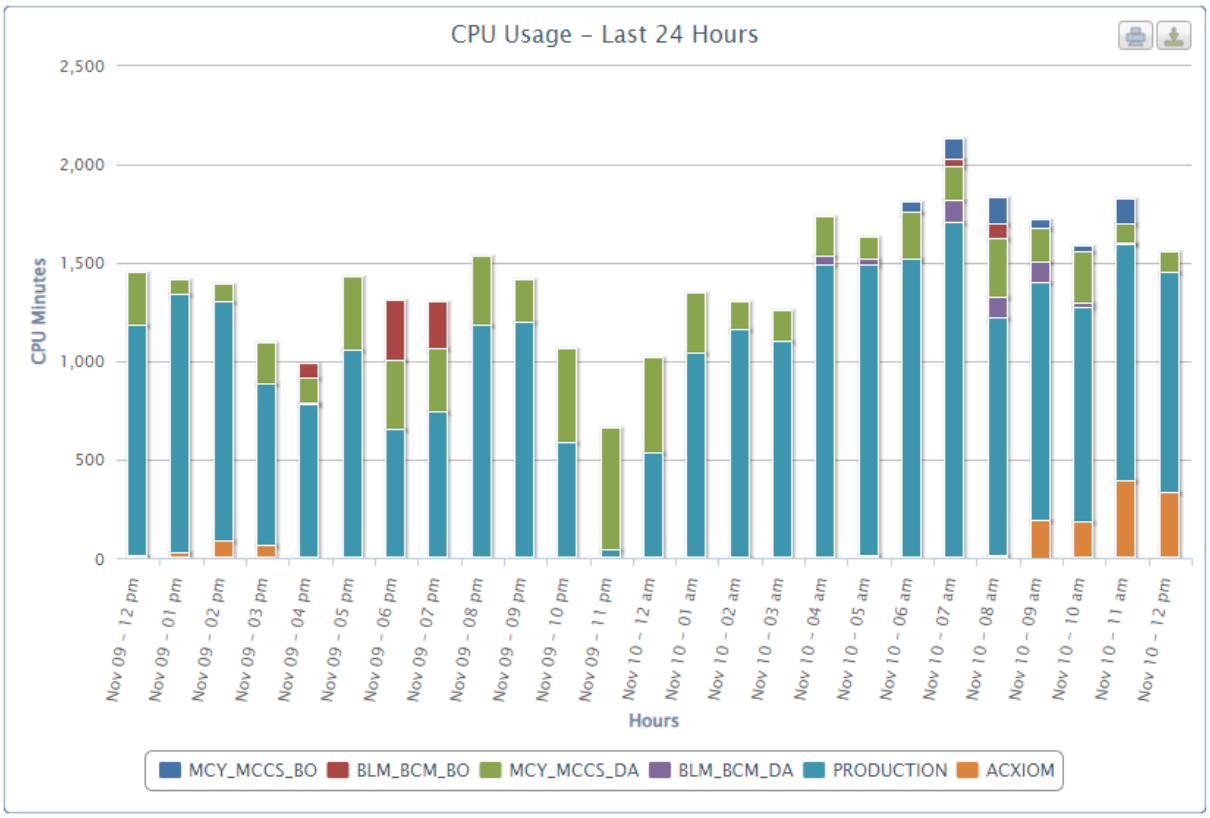
Using JavaScript and PHP, we used the arrays to load the data into the proper sections of the graph on the webpage. We also used JavaScript to create the names for the axis's and title.

JavaScript and PHP that implements arrays into webpage

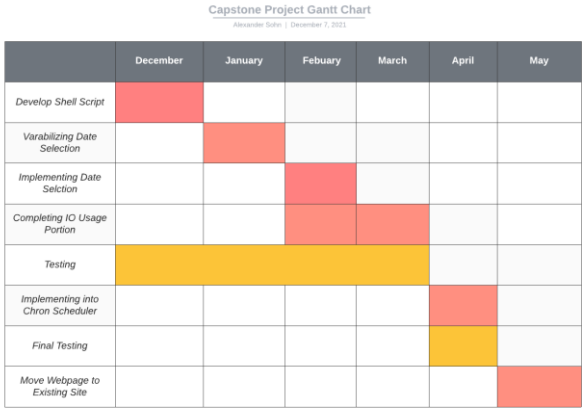
```
}, {
  name: 'PRODUCTION',
  data: [<?php
      $production_data = "";
      foreach($production_array as &$production_queries)
      {
        $production_data .= $production_queries . ",";
      }
      // strip last comma and echo
      $production_data = substr($production_data, 0, -1);
      echo $production_data;
      ?>
      ]
}, {
```

Past State Final

Finally, this is a screenshot of our current graph. As of now, we only have the CPU graph displayed, as getting the IO graph to work is simply changing the SQL a tad. For the CPU Usage, we have it displaying from Noon to Noon, this way our overnight production(the most important processing time) is shown without getting cut off.



Project Gantt Chart



Testing Methods Plan

We have already tested the current state, so before future state goals are implemented, we will test thoroughly. As we begin to reach the final states of the PHP, JavaScript, SQL, and Shell Script that we have written, we will send individual rough drafts to be looked over by our Solution Architect: Scott Travis. Once we start to put all the pieces, Scott will also test and try to break our webpage, to make sure it functions properly. After Scott is done testing all of the pieces and the prototypes are turned into final product, we will implement our solution into the existing tool website, where Scott will test for bugs and performance. When bugs or problems are found, Scott will reach out and they will be fixed and tested once again. This process will repeat until the webpage works smoothly; at which time we will announce to the rest of the team of the new tool added to the website.

Future State Plan

In the future, we will have 2 graphs one above the other, containing CPU Usage and IO Usage. To the left of these graphs, there will be a drop down or list of dates (EX: 12-NOV-21 / 12-NOV-21) in which a user can click on and update the graphs data. In regards to the shell script, we are going to implement it to run every 3 hours using a chron scheduler (This will keep the data up to date). The main .sql file containing creation of the table will work like this: A wip table will be built, this wip table will then be renamed into the main table, and the old table will be deleted. Other than this, pulling the dates that are selected will have to be worked on and figured out (going to variablize the selection on the webpage to be input into the query and update what data is pulled into the arrays).

Final State vs. The Plan

We were able to successfully implement 2 graphs, one containing CPU Usage and the other IO Usage, as well as a list of dates which a user can click on and update the page. We ended up having a shell script that runs every 4 hours using a cron scheduler, starting at 8:01 am, 12:01pm, etc. The tables are recreated by dropping the current table, creating a wip table, and then renaming the wip table to be the main table. We were able to figure out how to update the webpage based on the date range that is chosen. When the selection is clicked, the rdate and rdate2 variables are set to the dates. These are then set to id and id2 which can be seen in the link of the page. We then use the PHP variable $_GET to assign these values to the variables $START_HR and $END_HR, which are used in the query that pulls data from the table based off these 2 date values. (Code for this process is labeled: on page .)

Design and Development Work Performed

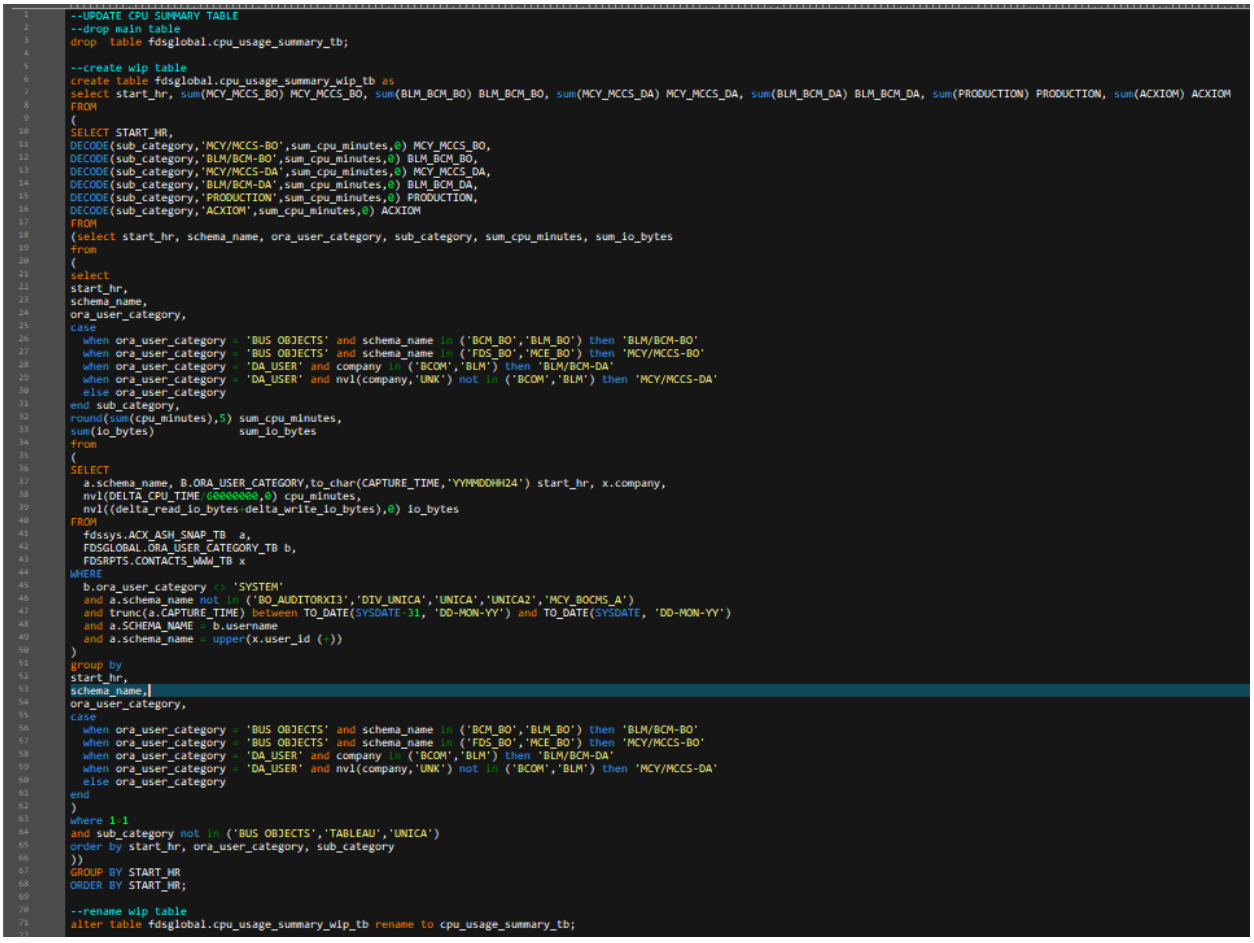
IT Infrastructure

- Hardware: Work Laptops, database server, and a webpage server
- Software: UltraEdit, Putty, PL/SQL Developer (most recent versions)
- Environments: We modified an old (no longer used) webpage that housed similar yet dated tools. We used this webpage to build and test our tool.

Data Architecture

We have 2 tables that feed our webpage data, named cpu_usage_summary_tb and io_usage_summary_tb. Both started in my schema, but we created production versions to use permanently. We also create a temporary wip table when recreating the tables with the most updated info. There were no privacy or security issues with the data, as there is no PII involved.

The following is an example of how these tables are built:



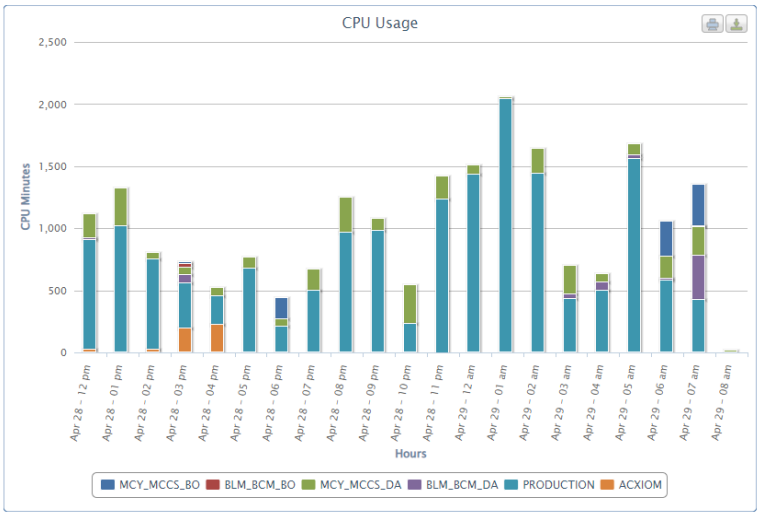
```
--UPDATE CPU SUMMARY TABLE
--drop main table
drop  table fdsglobal.cpu_usage_summary_tb;

--create wip table
create table fdsglobal.cpu_usage_summary_wip_tb as
select start_hr, sum(MCY_MCCS_BO) MCY_MCCS_BO, sum(BLM_BCM_BO) BLM_BCM_BO, sum(MCY_MCCS_DA) MCY_MCCS_DA, sum(BLM_BCM_DA) BLM_BCM_DA, sum(PRODUCTION) PRODUCTION, sum(ACXIOM) ACXIOM
FROM
(
SELECT START_HR,
DECODE(sub_category,'MCY/MCCS-BO',sum_cpu_minutes,0) MCY_MCCS_BO,
DECODE(sub_category,'BLM/BCM-BO',sum_cpu_minutes,0) BLM_BCM_BO,
DECODE(sub_category,'MCY/MCCS-DA',sum_cpu_minutes,0) MCY_MCCS_DA,
DECODE(sub_category,'BLM/BCM-DA',sum_cpu_minutes,0) BLM_BCM_DA,
DECODE(sub_category,'PRODUCTION',sum_cpu_minutes,0) PRODUCTION,
DECODE(sub_category,'ACXIOM',sum_cpu_minutes,0) ACXIOM
FROM
(select start_hr, schema_name, ora_user_category, sub_category, sum_cpu_minutes, sum_io_bytes
from
(
select
start_hr,
schema_name,
ora_user_category,
case
  when ora_user_category = 'BUS OBJECTS' and schema_name in ('BCM_BO','BLM_BO') then 'BLM/BCM-BO'
  when ora_user_category = 'BUS OBJECTS' and schema_name in ('FDS_BO','MCE_BO') then 'MCY/MCCS-BO'
  when ora_user_category = 'DA_USER' and company in ('BCOM','BLM') then 'BLM/BCM-DA'
  when ora_user_category = 'DA_USER' and nvl(company,'UNK') not in ('BCOM','BLM') then 'MCY/MCCS-DA'
  else ora_user_category
end sub_category,
round(sum(cpu_minutes),5) sum_cpu_minutes,
sum(io_bytes)             sum_io_bytes
from
(
SELECT
  a.schema_name, B.ORA_USER_CATEGORY,to_char(CAPTURE_TIME,'YYMMDDHH24') start_hr, x.company,
  nvl(DELTA_CPU_TIME/60000000,0) cpu_minutes,
  nvl((delta_read_io_bytes+delta_write_io_bytes),0) io_bytes
FROM
  fdssys.ACX_ASH_SNAP_TB  a,
  FDSGLOBAL.ORA_USER_CATEGORY_TB b,
  FDSRPTS.CONTACTS_WWW_TB x
WHERE
  b.ora_user_category <> 'SYSTEM'
  and a.schema_name not in ('BO_AUDITORXI3','DIV_UNICA','UNICA','UNICA2','MCY_BOCMS_A')
  and trunc(a.CAPTURE_TIME) between TO_DATE(SYSDATE-31, 'DD-MON-YY') and TO_DATE(SYSDATE, 'DD-MON-YY')
  and a.SCHEMA_NAME = b.username
  and a.schema_name = upper(x.user_id (+))
)
group by
start_hr,
schema_name,
ora_user_category,
case
  when ora_user_category = 'BUS OBJECTS' and schema_name in ('BCM_BO','BLM_BO') then 'BLM/BCM-BO'
  when ora_user_category = 'BUS OBJECTS' and schema_name in ('FDS_BO','MCE_BO') then 'MCY/MCCS-BO'
  when ora_user_category = 'DA_USER' and company in ('BCOM','BLM') then 'BLM/BCM-DA'
  when ora_user_category = 'DA_USER' and nvl(company,'UNK') not in ('BCOM','BLM') then 'MCY/MCCS-DA'
  else ora_user_category
end
)
where 1=1
and sub_category not in ('BUS OBJECTS','TABLEAU','UNICA')
order by start_hr, ora_user_category, sub_category
))
GROUP BY START_HR
ORDER BY START_HR;

--rename wip table
alter table fdsglobal.cpu_usage_summary_wip_tb rename to cpu_usage_summary_tb;
```

Webpage Interface

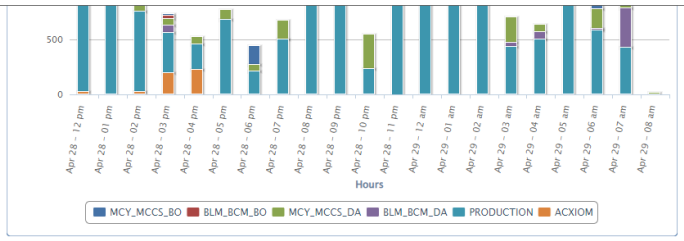
When the user loads up the webpage, the most recent data since noon the day before or of will be shown. When a user selects one of the links on the left side of the page, the page will refresh and populate the data with the date range selected. The graph is also dynamic. A user can select/deselect groups at the bottom of the graph to make them not be shown on the graph. For instance, if a user only wants to see production (the blue bar), they can deselect the other groups and only show the blue bars. If the user hovers over the bars of data, it will reveal the exact number of CPU Minutes or I/O gigabytes used.

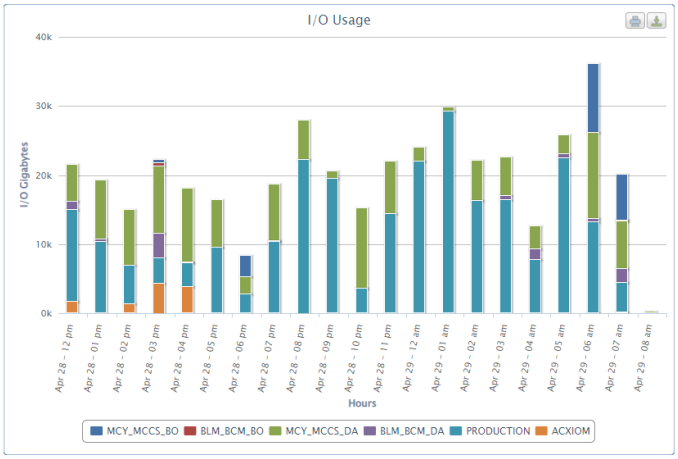
The following are screenshots of the webpage:

START	END	CLICK
29-MAR-2022	30-MAR-2022	Here
30-MAR-2022	31-MAR-2022	Here
31-MAR-2022	01-APR-2022	Here
01-APR-2022	02-APR-2022	Here
02-APR-2022	03-APR-2022	Here
03-APR-2022	04-APR-2022	Here
04-APR-2022	05-APR-2022	Here
05-APR-2022	06-APR-2022	Here
06-APR-2022	07-APR-2022	Here
07-APR-2022	08-APR-2022	Here
08-APR-2022	09-APR-2022	Here
09-APR-2022	10-APR-2022	Here
10-APR-2022	11-APR-2022	Here
11-APR-2022	12-APR-2022	Here
12-APR-2022	13-APR-2022	Here
13-APR-2022	14-APR-2022	Here
14-APR-2022	15-APR-2022	Here
15-APR-2022	16-APR-2022	Here
16-APR-2022	17-APR-2022	Here
17-APR-2022	18-APR-2022	Here
18-APR-2022	19-APR-2022	Here
19-APR-2022	20-APR-2022	Here
20-APR-2022	21-APR-2022	Here
21-APR-2022	22-APR-2022	Here
22-APR-2022	23-APR-2022	Here
23-APR-2022	24-APR-2022	Here
24-APR-2022	25-APR-2022	Here
25-APR-2022	26-APR-2022	Here
26-APR-2022	27-APR-2022	Here
27-APR-2022	28-APR-2022	Here
28-APR-2022	29-APR-2022	Here



15-APR-2022	16-APR-2022	Here
16-APR-2022	17-APR-2022	Here
17-APR-2022	18-APR-2022	Here
18-APR-2022	19-APR-2022	Here
19-APR-2022	20-APR-2022	Here
20-APR-2022	21-APR-2022	Here
21-APR-2022	22-APR-2022	Here
22-APR-2022	23-APR-2022	Here
23-APR-2022	24-APR-2022	Here
24-APR-2022	25-APR-2022	Here
25-APR-2022	26-APR-2022	Here
26-APR-2022	27-APR-2022	Here
27-APR-2022	28-APR-2022	Here
28-APR-2022	29-APR-2022	Here





Code

There is some code not shown due to the sensitivity of it, including the code for the webpage that houses are tool. Also not shown is the entry into our cron scheduler, but this is the format we used: 1 */4 * * *. This indicates that at minute 1 past every 4th hour, the shell script will run.

The following are screenshots of the SQL, PHP, HTML, CSS, and JavaScript that was used to build the webpage:

Filling arrays with data from table:

```php
<?php

error_reporting(0);

$stid = oci_parse($c,"select START_TIME, END_TIME, to_char(START_TIME, 'YYMMDDHH24') as Q_TIME, to_char(END_TIME, 'YYMMDDHH24') as QE_TIME,
CLICK
from
(
select START_TIME, START_TIME + 1 as END_TIME, CLICK
from
(select to_date(start_hr, 'YYMMDDHH24') as START_TIME, 'Here' as click FROM fdsglobal.cpu_usage_summary_tb
where start_hr like '%12'))"); // fetch data from database
$r = oci_execute($stid);

// Fetch each row in an associative array
print '<div class="dates" style = "margin-left: 100px; margin-top: 25px;"> <table width=200 border=1 style="border:2px solid black; float: left;">

<tr>
<th>START</th>
<th>END</th>
<th>CLICK</th>
</tr>';


while ($row = oci_fetch_array($stid)) {
  $rdate = $row['Q_TIME'];
  $rdate2 = $row['QE_TIME'];

   print '<tr id="' $rdate.'">';

   print '        <td class=td>' $row['START_TIME'] '</td>' "\n";
    print '        <td class=td>' $row['END_TIME'] '</td>' "\n";
     print '<td  <td class=td style="font-weight:bold">';
    print '        <a href="prod_dashboard.php?pg=querymetrics&pserv=mcyprod&id=' $rdate '&id2=' $rdate2.'" style="color:#002B88;cursor:pointer">' $row['CLICK'] "</a></td>\n";

    print '      </tr>' "\n";


}
print '</table></div>';


$START_HR = $_GET['id'];
$END_HR = $_GET['id2'];


if (is_null ($START_HR) ){
  $database_queries_query = oci_parse($c,"select * from fdsglobal.cpu_usage_summary_tb
where start_hr between to_char(sysdate-1, 'YYMMDDHH24') and to_char(sysdate, 'YYMMDDHH24')");
}
else {
$database_queries_query = oci_parse($c,"select * from fdsglobal.cpu_usage_summary_tb
where start_hr between ".$START_HR." and ".$END_HR."");
}
```

```php
oci_execute($database_queries_query, OCI_DEFAULT);

//ARRAYS FOR SUBCATEGORIES
$mcy_mccs_bo_array = array();
$blm_bcm_bo_array = array();
$mcy_mccs_da_array = array();
$blm_bcm_da_array = array();
$production_array = array();
$acxiom_array = array();
$array_index = 0;

// build arrays with results
while ($database_queries_result = oci_fetch_array($database_queries_query))
{
    $mcy_mccs_bo_array[$array_index] = $database_queries_result['MCY_MCCS_BO'];
    $blm_bcm_bo_array[$array_index] = $database_queries_result['BLM_BCM_BO'];
    $mcy_mccs_da_array[$array_index] = $database_queries_result['MCY_MCCS_DA'];
    $blm_bcm_da_array[$array_index] = $database_queries_result['BLM_BCM_DA'];
    $production_array[$array_index] = $database_queries_result['PRODUCTION'];
    $acxiom_array[$array_index] = $database_queries_result['ACXIOM'];
  $array_index++;
}



// get the labels for the database queries query
if (is_null ($START_HR) ){
$db_queries_lbl_query = oci_parse($c,"select TO_CHAR(TRUNC(labels,'HH24'),'Mon DD - HH am') HR from
(select to_date(start_hr, 'YYMMDDHH24') labels from fdsglobal.cpu_usage_summary_tb
where start_hr between to_char(sysdate-1, 'YYMMDDHH24') and to_char(sysdate, 'YYMMDDHH24')
)");
}
else {
$db_queries_lbl_query = oci_parse($c,"select TO_CHAR(TRUNC(labels,'HH24'),'Mon DD - HH am') HR from
(select to_date(start_hr, 'YYMMDDHH24') labels from fdsglobal.cpu_usage_summary_tb
where start_hr between ".$START_HR." and ".$END_HR.")");
}



oci_execute($db_queries_lbl_query, OCI_DEFAULT);

$lbl_array = array();
$lbl_index = 0;
while ($db_queries_lbl_result = oci_fetch_array($db_queries_lbl_query))
{
    $lbl_array[$lbl_index] = $db_queries_lbl_result['HR'];
  $lbl_index++;
}
```

Shell Script that updates the tables:

```
#!/bin/ksh

#####################################################################
# Updates CPU_USAGE_SUMMARY_TB and IO_USAGE_SUMMARY_TB
#####################################################################

main(){

    orauser=
    orapw=`/home/        /bin/getorapw ${orauser}`


    sqlplus ${orauser}/${orapw} @/home/        /bin/upt_cpu_io_usage_summary_tb.sql



    }
    . /home/        bin/set_fds_env.sh
    . /home/        bin/setoraenv.sh

    main $*

```

Building the Graph:

```
<!-- CPU Usage Summary chart -->
<script type="text/javascript">

  var dbquerieschart;
  $(document).ready(function() {
    dbquerieschart = new Highcharts.Chart({

      chart: {
        renderTo: 'dbcpucontainer',
        defaultSeriesType: 'column'
      },

      title: {
        text: 'CPU Usage'
      },

      xAxis: {
        categories: [
              <?php
              $category_string = "";
              foreach($lbl_array as &$label)
              {
                $category_string .= "'" . $label . "',";
              }
              // strip last comma and echo
              $category_string = substr($category_string, 0, -1);
              echo $category_string;
              ?>
              ],
              labels: {
                rotation: 280,
                align: 'right'
              },
                title: {
                text: 'Hours',
                margin: 15
              }
      },

      yAxis: {
        allowDecimals: false,
        min: 0,
        title: {
          text: 'CPU Minutes'
        }
      },
```

```
      series: [{
      name: 'MCY_MCCS_BO',
      data: [<?php
          $mcy_mccs_bo_data = "";
          foreach($mcy_mccs_bo_array as &$mcy_mccs_bo_queries)
          {
            $mcy_mccs_bo_data .= $mcy_mccs_bo_queries . ",";
          }
          // strip last comma and echo
          $mcy_mccs_bo_data = substr($mcy_mccs_bo_data, 0, -1);
          echo $mcy_mccs_bo_data;
          ?>
          ]
    }, {
      name: 'BLM_BCM_BO',
      data: [<?php
          $blm_bcm_bo_data = "";
          foreach($blm_bcm_bo_array as &$blm_bcm_bo_queries)
          {
            $blm_bcm_bo_data .= $blm_bcm_bo_queries . ",";
          }
          // strip last comma and echo
          $blm_bcm_bo_data = substr($blm_bcm_bo_data, 0, -1);
          echo $blm_bcm_bo_data;
          ?>
          ]
    }, {
      name: 'MCY_MCCS_DA',
      data: [<?php
          $mcy_mccs_da_data = "";
          foreach($mcy_mccs_da_array as &$mcy_mccs_da_queries)
          {
            $mcy_mccs_da_data .= $mcy_mccs_da_queries . ",";
          }
          // strip last comma and echo
          $mcy_mccs_da_data = substr($mcy_mccs_da_data, 0, -1);
          echo $mcy_mccs_da_data;
          ?>
          ]
    }, {
```

Testing Methods

Throughout the entirety of my project, Cade and I unit tested individual components of our code to make sure it would run as expected. We then integrated our code together and tested it to make sure all interactions would run seamlessly. Performance and Usability testing were also performed with help from Scott, which allowed us to change our performance from refreshing the page taking 2 minutes to less than a second.

Teamwork

There was a total of 5 people that contributed in someway to this project. Shane was the stakeholder and brought this project to life. Along with him, Scott and Kevin helped test, monitor, provide feedback, etc. throughout the span of the project. Cade worked alongside me to help build the webpage and provided feedback on the direction I was taking. He also was able to troubleshoot any errors or problems that were being ran into. I would meet with Cade at least once a week, and I would meet with Shane every 2 weeks to discuss the projects progress and make changes if needed. When it came to testing and implementing the final version, Scott helped teach us about the cron scheduler and how to make production versions of our code and tables. The team environment was amazing, and everyone came together to help if problems arose.

Support and Integration

This project aligns with the direction my team is taking. This includes making and improving current tools we use to maintain the system and a happy relationship with our client. To support our tool in the long run, the team needs to make sure the shell script that updates the data used in the webpage is ran accordingly with the cron scheduler. If the server were to go down during the scheduling of this run, it would have to be ran manually, or the team would have to wait 4 hours to the next scheduled run.

Closing Thoughts

Lessons Learned

When I started this project, I had no clue what direction the project would take or the complexities behind it. I had no experience using shell scripts, PHP, a cron scheduler, and very little JavaScript experience. Using an old webpage with similar structure to how our webpage was being designed, I learned looking at its code how to use SQL, PHP, and JavaScript to make a

dynamic webpage that pulled data from a table. I also learned how to properly document and test during the duration of the project.

Future Changes

Some futures changes that could be made is optimizing the webpage for better accessibility and how it acts dynamically when the screen size is changed. This includes the placement of the table, graphs, and how they move together when sized differently. The CSS could also be changed as its not as sharp as I would like it to look.

Appendices

Around 225 hours were put into this project. This includes design, research, testing, and writing this report. My co-worker and mentor Cade worked alongside me on this project and helped direct its path.