# Homework 1: DDL Processing for a Parallel DBMS

- Homework 1 Page
- Installation
  - Install Docker Containers
  - Setup a Docker Container
  - Setup Python
  - How to Install Homework 1 Program
  - How to Run Homework 1 Program
- Overview
  - Directory Structure
  - File and Program Descriptions
  - Server Files
  - Configuration Files
  - Client Program
  - Expected Output and Error Conditions
  - Expected Output
  - Error Conditions
- Cheat Sheets
  - Docker Cheat Sheet
  - Linux Cheat Sheet

# Installation

## Install Docker Containers

First, download the Docker Community Edition (CE) for the Desktop from their download page.

Secondly, follow their instructions at their Getting Started Page.

If you want hands-on tutorials or would like to use their training videos and online playground, please feel free to do so.

## Setup a Docker Container

First, open up a terminal or command prompt window and cd to the designated Docker directory:

```
cd Desktop/Docker
```

After you're in your designated Docker directory, start your own container. In this case, I named my container "myContainer" and started it up with Ubuntu:

```
docker run -it --name=[myContainer] ubuntu
```

Since the base Ubuntu image only has the bare minimal packages installed, we will need to install some more packages:

```
apt-get -y update
apt-get -y install iputils-ping
apt-get -y install iproute
apt-get -y install dnsutils
```

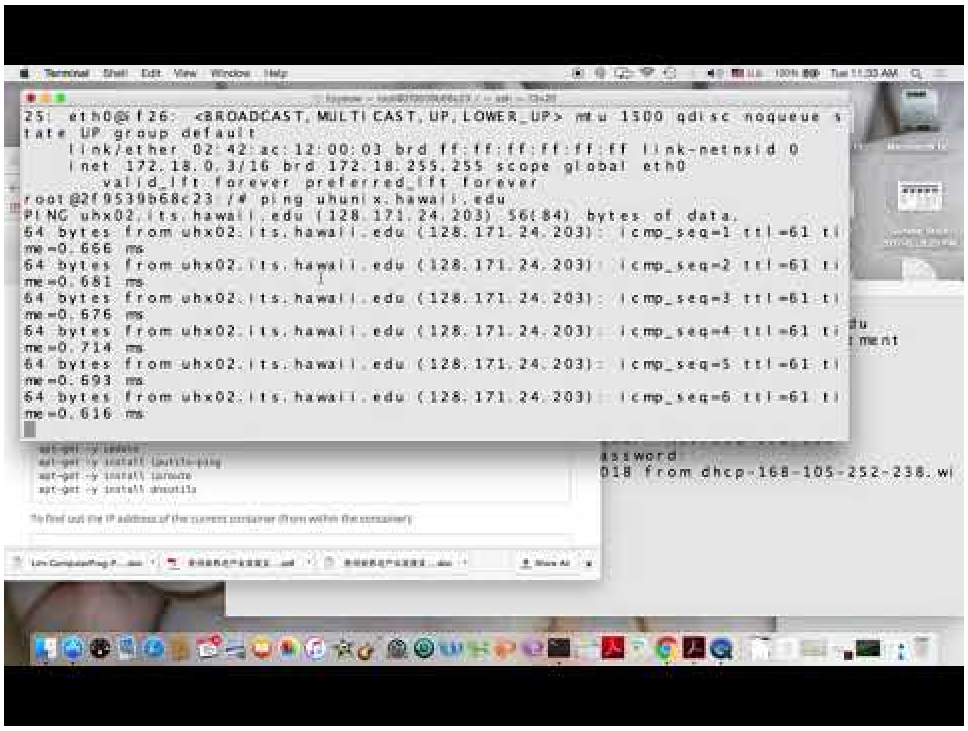
You can find the ip address of your current container by doing ip a. For example, my container's ip is 172.17.0.2.:

```
ip a
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1
    link/ipip 0.0.0.0 brd 0.0.0.0
3: ip6tnl0@NONE: <NOARP> mtu 1452 qdisc noop state DOWN group default qlen 1
    link/tunnel6 :: brd ::
6: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

You can also ping the address of your container or other containers that you have created by using the ping:

```
ping 172.18.0.3
```

Professor Lipyeow Lim provides us with a very good demonstration of setting up a docker container and pinging: 

```
25: eth0@if26: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue s
tate UP group default
    link/ether 02:42:ac:12:00:03 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.18.0.3/16 brd 172.18.255.255 scope global eth0
       valid_lft forever preferred_lft forever
root@2f9539b68c23 /# ping uhunix.hawaii.edu
PING uhx02.its.hawaii.edu (128.171.24.203) 56(84) bytes of data.
64 bytes from uhx02.its.hawaii.edu (128.171.24.203): icmp_seq=1 ttl=61 ti
me=0.666 ms
64 bytes from uhx02.its.hawaii.edu (128.171.24.203): icmp_seq=2 ttl=61 ti
me=0.681 ms
64 bytes from uhx02.its.hawaii.edu (128.171.24.203): icmp_seq=3 ttl=61 ti
me=0.676 ms
64 bytes from uhx02.its.hawaii.edu (128.171.24.203): icmp_seq=4 ttl=61 ti
me=0.714 ms
64 bytes from uhx02.its.hawaii.edu (128.171.24.203): icmp_seq=5 ttl=61 ti
me=0.693 ms
64 bytes from uhx02.its.hawaii.edu (128.171.24.203): icmp_seq=6 ttl=61 ti
me=0.616 ms
```

## Setup Python

Now that we've installed the packages we need, let's also install Python3, package installer, vim, and sqlite3:

```
apt-get -y install python3
apt-get -y install python3-pip
apt-get -y install vim
apt-get -y install sqlite3
```

Since it is not a good practice to work in root, let's create a user account:

```
adduser [username]
```

Make sure to follow the onscreen instructions!

```
su [username]
cd
```

## How to Install Homework 1 Program

Leave the terminal or command prompt open and go to my github page. Download the zip containing my homework 1 files and store / unzip it into the designated docker directory.

Now start up a new terminal or command prompt tab or window and go to your designated docker directory, and copy the files into your container. In my case:

```
cd Desktop/Docker
chmod -R 0777 hw1-master
docker cp /Users/SaeHyunSong/Desktop/Docker/hw1-master [Container Name]:/home/[user
name]/
```

Now go back to your container and cd into the hw1-master directory and voila we're all done with the installation process!

## How to Run Homework 1 Program

Go to the catalog directory and start up the server, default hostname: 127.0.0.10 and port: 5000:

```
cd catalog
python3 parDBd.py &
```

Go to the node1 directory and start up the server with the commandline arguments that match the cluster.cfg file's hostname and port number:

```
cd node1
python3 parDBd.py 127.0.0.2 5000 &
```

Go to the node2 directory and start up the server with the commandline arguments that match the cluster.cfg file's hostname and port number:

```
cd node2
python3 parDBd.py 127.0.0.3 5000 &
```

Go back to the hw1 directory and run the shell script run.sh:

```
./run.sh cluster.cfg books.sql
```

You should then see the expected output.

If you want to do this again, make sure to delete the tables in the mydb1 and mydb2 databases by doing:

```
cd node1
sqlite3 mydb1
DROP TABLE BOOKS;
.exit

cd ../node2
sqlite3 mydb2
DROP TABLE BOOKS;
```

```
.exit
```

Then run everything again from the beginning. If you get any errors, refer to the Error Conditions Section.

# Overview

## Directory Structure

The top-level directory structure contains:

```
catalog/           # Holds the sqlite3 database mycatdb and the server program.
  mycatdb          # The sqlite3 database called mycatdb.
  parDBd.py        # The server program for the catalog node.

node1/             # Holds the sqlite3 database mydb1 and the server program to
simulate a different computer.
  mydb1            # The sqlite3 database called mydb1.
  parDBd.py        # The server program for the node1 node.

node2/             # Holds the sqlite3 database mydb2 and the server program to
simulate a different computer.
  mydb2            # The sqlite3 database called mydb2.
  parDBd.py        # The server program for the node2 node.

README.md          # Contains information about installation and files.
cluster.cfg        # A configuration file that contains information about the nodes.
books.sql          # Contains a DDL command which is used to run on the node's
database.
run.sh             # A shell script to run runDDL.py with two commandline arguments
cluster.cfg and books.sql
runDDL.py          # Contains the main source code to parse config file and sql file
and send info to the node servers.
```

## File and Program Descriptions

### Server Files

There are three server files one for each node called parDBd.py along with their own sqlite3 database:

```
catalog/           # Holds the sqlite3 database mycatdb and the server program.
  mycatdb          # The sqlite3 database called mycatdb.
  parDBd.py        # The server program for the catalog node.

node1/             # Holds the sqlite3 database mydb1 and the server program.
  mydb1            # The sqlite3 database called mydb1.
  parDBd.py        # The server program for the node1 node.
```

```
node2/              # Holds the sqlite3 database mydb2 and the server program.
  mydb2             # The sqlite3 database called mydb2.
  parDBd.py         # The server program for the node2 node.
```

The node1 and node2 directory has the same server program but different database names:

```
node1/              # Holds the sqlite3 database mydb1 and the server program.
  mydb1             # The sqlite3 database called mydb1.
  parDBd.py         # The server program for the node1 node.

node2/              # Holds the sqlite3 database mydb2 and the server program.
  mydb2             # The sqlite3 database called mydb2.
  parDBd.py         # The server program for the node2 node.
```

The parDBd.py program for node1/ and node2/ contains:

```
# Importing all the necessary libraries.
import socket

import sqlite3
from sqlite3 import Error

import sys
from sys import argv

# A main function that takes in two commandline arguments.
def Main(argv):
    host = argv[1];
    port = argv[2];
    #host = "127.0.0.2";
    #port = 5000;

    # Store data received into datas array.
    datas = [];

    mySocket = socket.socket()
    mySocket.bind((str(host),int(port)))

    mySocket.listen(1)
    conn, addr = mySocket.accept()
    # print ("Server: Connection from " + str(addr))
    data = conn.recv(1024).decode()
    # print("DATA:", data);
    if not data:
        return
    # print ("Server: recv " + str(data));
    datas.append(data.split("$")[0]);
    datas.append(data.split("$")[1]);

    # Connect to sqlite database in node1 directory and execute DDL command.
    try:
        condb = sqlite3.connect("../node2" + datas[0]);
        # print(sqlite3.version);
        cur = condb.cursor();
        cur.execute(datas[1]);
        message = "./books.sql success.$" + host + ":" +  str(port) + datas[0];
        condb.commit();
        conn.send(message.encode());
    # If there is an error, send a message back to client that it was a failure.
    except Error as e:
        # print(e);
        message = "./books.sql failure.$" + host + ":" + str(port) + datas[0];
        conn.send(message.encode());
    # After everything, finally close the db and the connection between client /
server.
    finally:
        condb.close();
        conn.close();
# Run main function with argv parameters (commandline arguments)
Main(argv);
```

What this server program is basically doing is that it takes in two commandline arguments: IP address or hostname, and the port number. It uses those command line arguments to generate a socket with that ip address and port number. The socket will then listen for any data sent to it. Using the data it has received, it will connect to it's sqlite3 database with the data containing the database name. Execute the DDL statement that was sent along with the database name. Which then it will send a message back to the client program that it was successful or not. Then it finally closes all the connections after everything has been done.

The catalog directory has a slightly different server program compared to the node1 and node2 servers:

```
catalog/          # Holds the sqlite3 database mycatdb and the server program.
  mycatdb         # The sqlite3 database called mycatdb.
  parDBd.py       # The server program for the catalog node.
```

The parDBd.py program for catalog/ contains:

```
# Import all necessary libraries.
import socket

import sqlite3
from sqlite3 import Error

# A Main function which listens for a message from the nodes to create catalog db
and update the db.
def Main():
    # Host / port initialized with constant values.
    host = "127.0.0.5"
    port = 5000

    # Messages received will be store in datas array.
    datas = [];
    mySocket = socket.socket()
    mySocket.bind((host,port))

    mySocket.listen(1)
    conn, addr = mySocket.accept()
    # print ("Server: Connection from " + str(addr))
    data = conn.recv(1024).decode()
    if not data:
        return
    # print ("Server: recv " + str(data));
    datap = data.split("$");
    # Connect to the mycatdb sqlite3 database and execute a create table / insert
DDL command.
    try:
        con = sqlite3.connect("../catalog/mycatdb");
        cur = con.cursor();
        cur.execute("CREATE TABLE IF NOT EXISTS DTABLES(tname char(32), nodedriver
char(64), nodeurl char(128), nodeuser char(16), nodepasswd char(16), partmtd int,
nodeid int, partcol char(32), partparam1 char(32), partparam2 char(32));");
        cur.execute("INSERT INTO DTABLES(tname, nodedriver, nodeurl, nodeuser,
nodepasswd, partmtd, nodeid, partcol, partparam1, partparam2) VALUES (" + "'" +
datap[1] + "'" + ", NULL, " + "'" + datap[2] + "'" + ", NULL, NULL, NULL, 1, NULL,
NULL, NULL);");
        con.commit();
        message = "catalog updated.";
        conn.send(message.encode());
    # If there is an error send back an error message to client.
    except Error as e:
        # print(e);
        message = e;
        conn.send(message.encode());
        con.close();
    # Finally, close all connections.
    finally:
        con.close();

if __name__ == '__main__':
    Main();
    Main();
```

What this server program does differently compared to the other server programs for node1 and node2 is that it receives from the client that the node1 or node2 was successful. Which then it creates the table called DTABLES it id does not already exist in the catalog database. After checking that, it will store the metadata about the DDL being executed in the catalog database. Then it sends the message back to the client that the catalog has been updated. The good thing about this is that the client will only send a message to the catalog server if node1 and node2 create tables were successful, saving time. It will be explained more when I am describing the client program runDDL.py.

## Configuration Files

There are two configuration files, books.sql and cluster.cfg:

```
cluster.cfg        # A configuration file that contains information about the nodes.
books.sql          # Contains a DDL command which is used to run on the node's
database.
```

The cluster.cfg file contains access information for each computer on the cluster such as the hostname, port, database name, and the number of nodes. This file will be parsed and the data will be used to send information to the cluster of node server programs.

```
catalog.hostname=127.0.0.1:5000/mycatdb

numnodes=2

node1.hostname=127.0.0.2:5000/mydb1

node2.hostname=127.0.0.3:5000/mydb2
```

The file book.sql contains the DDL terminated by a semi-colon to be executed on the node server:

```
CREATE TABLE BOOKS(isbn char(14), title char(80), price decimal);
```

## Client Program

The client program called runDDL.py contains:

```
# Importing socket library to do the socket connections between server and client.
import socket

# Importing sqlite3 library to do sqlite3 functions.
import sqlite3
from sqlite3 import Error

# Importing sys library to taking in commandline arguments.
import sys
from sys import argv

def runDDL(argv):
    # Read from the cluster.cfg file and store it into an array called data.
    data = [];
    configFile = open(argv[1], "r");
    data = configFile.read().strip().replace("\n",";").split(';');
    configFile.close();

    # Read from the books.sql file and store the DDL into ddlCommands array.
    data = list(filter(('').__ne__, data));
    ddlCommands = [];
    ddlFile = open(argv[2], "r");
    ddlCommands= ddlFile.read().strip().replace("\n","").split(';');
    ddlFile.close();

    # Initialize variables to store data in.
    url = '';
    hostname = '';
    port = '';
    db = '';
    numnodes = 0;
    nodes = [];

    # Parse the data in the cluster.cfg file and store them into their respective
variables
    for d in data:
        if d.strip():
            temp = d.strip().split("=");
            if temp[0].find("catalog") > -1:
                if temp[0].find("hostname") > -1:
                    url = temp[1];
                    hostname = temp[1].split("/")[0].split(":")[0];
                    port = temp[1].split("/")[0].split(":")[1];
                    db = temp[1].split("/")[1];
                    nodes.append(Node(url, hostname, port, db));
            if temp[0].find("node") > -1:
                if temp[0].find("hostname") > -1:
                    url = temp[1];
                    hostname = temp[1].split("/")[0].split(":")[0];
                    port = temp[1].split("/")[0].split(":")[1];
                    db = temp[1].split("/")[1];
                    nodes.append(Node(url, hostname, port, db));

    # Get the number of nodes and find the tablename / message.
    numnodes = len(nodes);
    tablename = ddlCommands[0].split("(")[0].split(" ")[2];
    message = nodes[1].url.split("/", 1)[1];

    # Loop through each node and send a message to each node's server.
    x = 1;
    while(x < numnodes):
        message = nodes[x].url.split("/", 1)[1];
        message = "/" + message + "$" + ddlCommands[0];
        mySocket = socket.socket();
        mySocket.connect((nodes[x].hostname, int(nodes[x].port)));
        mySocket.send(message.encode())
        received = mySocket.recv(1024).decode();
        receivedp = received.split("$");
        mySocket.close();
        # Receive a message back from the server, if success send a command to the
catalog server to create a table / store data.
        if receivedp[0] == "./books.sql success.":
            print("[" + nodes[x].url + "]: " + receivedp[0]);
            message2 = receivedp[0] + "$" + tablename + "$" + nodes[x].url;
            mySocket2 = socket.socket();
            mySocket2.connect((nodes[0].hostname, int(nodes[0].port)));
            mySocket2.send(message2.encode());
            received2 = mySocket2.recv(1024).decode();
            # If the catalog has been updated print that it has been.
            if received2 == "catalog updated.":
                print("[" + nodes[0].url + "]: " + received2);
            else:
                print("[" + nodes[0].url + "]: " + received2);
            mySocket2.close();
        # If not a success, print that the catalog has not been updated.
        else:
            print("[" + nodes[x].url + "]: " + receivedp[0]);
            print("[" + nodes[0].url + "]: catalog not updated.");
        x += 1;

# A class called node containing the url, hostname, port, and db name of the node.
class Node:
    def __init__(self, url, hostname, port, db):
        self.url = url;
        self.hostname = hostname;
        self.port = port;
        self.db = db;
    def displayNode(self):
        print("URL:", self.url, "HOSTNAME:", self.hostname, "PORT:", self.port,
"DB:", self.db);

# Run the function runDDL with 2 commandline arguments.
runDDL(argv);
```

The runDDL program takes in two commandline arguments which are cluster.cfg and books.sql. It will then parse the books.sql file and insert it into an array called ddlCommands. It also will parse the cluster.cfg file and store them into an array called data. Which then I create a loop for each data to split them and place each value in their respective variables and append that data into a Node class. Then I store each instance of the Node class that was created into the nodes array. Finally, I loop through each each node and send the data that was parsed from the cluster.cfg and books.sql file to the node servers and print out the message I receive from the server of whether or not it was successful. If it was successful, it will send that data to the catalog node to update the database there with the metadata. Otherwise, it will not send anything and just print out that the catalog was not updated.

## Expected Output and Error Conditions

### Expected Output

The expected output should be:

```
[127.0.0.2:5000/mydb1]: ./books.sql success.
[127.0.0.1:5000/mycatdb]: catalog updated.
[127.0.0.3:5000/mydb2]: ./books.sql success.
[127.0.0.1:5000/mycatdb]: catalog updated.
```

### Error Conditions

An error condition would be if the ip address or hostname is inputted wrong / doesn't match the cluster.cfg file when using commandline arguments, it will give a connection refused error:

```
Traceback (most recent call last):
  File "runDDL.py", line 100, in <module>
    runDDL(argv);
  File "runDDL.py", line 64, in runDDL
    mySocket.connect((nodes[x].hostname, int(nodes[x].port)));
ConnectionRefusedError: [Errno 111] Connection refused
```

Another error condition would be if the server is not finished shutting down, it will say that the server ip address is already in use. I'd suggest waiting about a minute to restart the server:

```
  File "parDBd.py", line 53, in <module>
    Main(argv);
  File "parDBd.py", line 21, in Main
    mySocket.bind((str(host),int(port)))
OSError: [Errno 98] Address already in use
```

Another error condition is if the table already exists in the node database and you try to execute the CREATE TABLE DDL statement again:

```
[127.0.0.2:5000/mydb1]: ./books.sql failure.
[127.0.0.1:5000/mycatdb]: catalog not updated.
[127.0.0.3:5000/mydb2]: ./books.sql failure.
[127.0.0.1:5000/mycatdb]: catalog not updated.
```

You can get around this by deleting the table in the node server database by doing:

```
sqlite3 [database name]
DROP TABLE [TABLE NAME];
.exit
```

# Cheat Sheets

## Docker Cheat Sheet

Here is a simple Docker Cheat Sheet provided by Docker in case you're unfamiliar with the commands.

## Linux Cheat Sheet

Here is a Linux/Unix Commands Reference in case you're unfamiliar with the system as we will be using a Docker Linux Container with Ubuntu.