Homework Package

Question 1: Application Running with SQL.

Application URL on pythonAnywhere : I deployed the application on pythonanywhere VM

http://deba.pythonanywhere.com

Source Code :

```
import sqlite3
from bottle import route, run, template, debug, request

@route('/')
@route('/index.html')
@route('/tasks')
def tasks():
    conn = sqlite3.connect('todo.db')
    c = conn.cursor()
    c.execute("SELECT id, task FROM todo WHERE status
LIKE '1'")
    result = c.fetchall()
    c.close()
    output = template('current_tasks', rows=result)
    return output

@route('/new', method='GET')
def new():
    return template('new_task')

@route('/new', method='POST')
def new_item():
    new = request.POST.task.strip()

    conn = sqlite3.connect('todo.db')
    c = conn.cursor()

    c.execute("INSERT INTO todo (task,status) VALUES (?,?)",
(new, 1))
    new_id = c.lastrowid

    conn.commit()
    c.close()

    return '<p>The new task was inserted into the database, the
ID is %s</p>' % new_id

    c = conn.cursor()
    c.execute("DELETE from todo WHERE id LIKE ?", (id,))
```

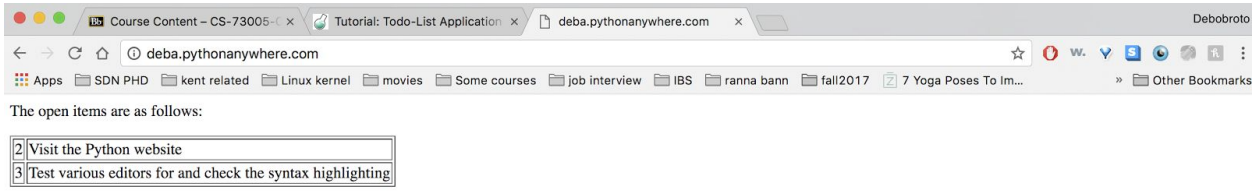
```
@route('/edit/<id>', method='GET')
def edit(id):
    conn = sqlite3.connect('todo.db')
    c = conn.cursor()
    c.execute("SELECT id, task, status FROM todo WHERE id
LIKE ?",(id,))
    result = c.fetchall()
    if len(result) == 0:
        return "Error: wrong number of results"
    result = result[0]
    id, text, status = result
    c.close()
    output = template('edit_task', id=id, text=text, status=status)
    return output

@route('/edit/<id>', method='POST')
def edit(id):
    updated_task = request.POST.task.strip()

    conn = sqlite3.connect('todo.db')
    c = conn.cursor()

    c.execute("UPDATE todo SET task = ? WHERE id LIKE ?",
(updated_task,id))
    new_id = c.lastrowid

    conn.commit()
    c.close()

@route('/complete/<id>', method='GET')
def complete(id):
    conn = sqlite3.connect('todo.db')
    c = conn.cursor()
    c.execute("UPDATE todo SET status = 0 WHERE id LIKE
?", (id,))

    conn.commit()
    c.close()
```

```
    conn.commit()
    c.close()

    return '<p>The new task was deleted from the database, the ID is %s</p>' % id



debug(True)
run(host='localhost', port=8080)
```

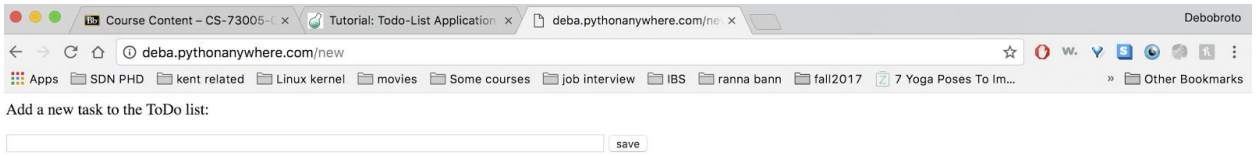
```
    return '<p>The new task was marked complete in the database, the ID is %s</p>' % id

@route('/delete/<id>', method='GET')
def delete(id):
    conn = sqlite3.connect('todo.db')
```

Screenshot

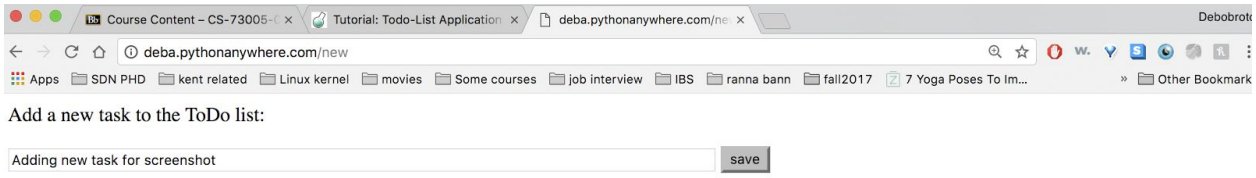
List of tasks :



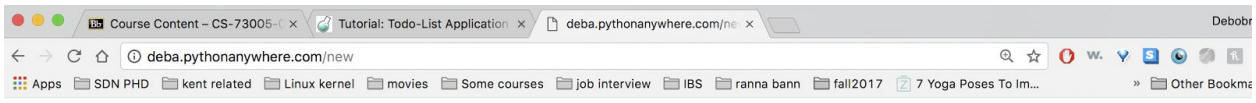
Add Task Form:



Input Task Details :



Task Added Successfully:

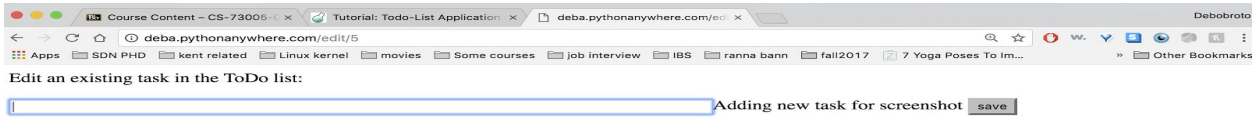


Updated Task List After New Task Added:

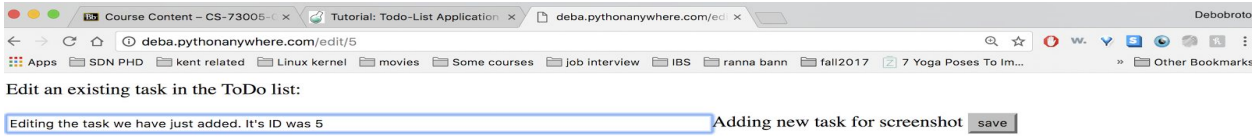
The open items are as follows:

2	Visit the Python website
3	Test various editors for and check the syntax highlighting
5	Adding new task for screenshot

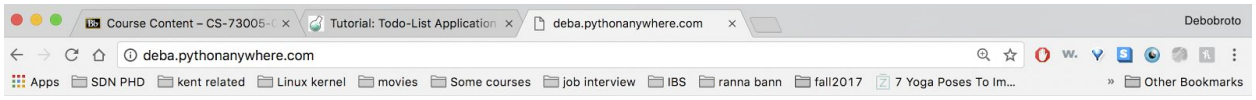
Edit Task Form:



Edit Task Details:



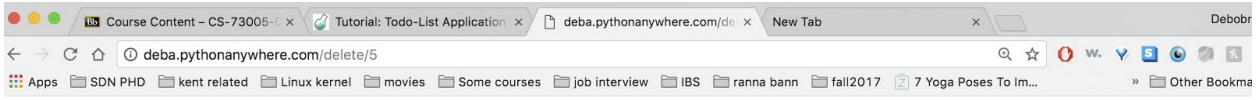
Task List After Task Updated:



The open items are as follows:

2	Visit the Python website
3	Test various editors for and check the syntax highlighting
5	Editing the task we have just added. It's ID was 5

Delete Task:



The new task was deleted from the database, the ID is 5

Question 2: Chinook Database query that involves join of 3 tables

Without custom Index:

Tables that are joined:

a) Album
b) Artist
c) Genre
d) Playlist
e) PlaylistTrack

Query:

```
SELECT Artist.Name, Genre.Name, COUNT(*)
FROM Playlist, PlaylistTrack,
Track, Album, Artist, Genre
WHERE Playlist.PlaylistId = PlaylistTrack.PlaylistId
AND PlaylistTrack.TrackId = Track.TrackId
AND Track.AlbumId = Album.AlbumId
AND Album.ArtistId = Artist.ArtistId
AND (Genre.Name='Pop' )
AND Genre.GenreID = Track.GenreID
GROUP BY Genre.Name, Artist.Name
```

What the query does:

It lists name of all the artists who sings “Pop” songs, and how many times their tracks have been listed in Track table.

With custom index:

New Index created is following:

Index:

```
CREATE INDEX Custom_index_Genre_Name ON Genre (
        Name
);
```

Advantage of this index: This index will make the query faster. Because, our previous query searches on genre Name. Now after adding index on that field, out query will be faster.

Tables that are joined:

f) Album
g) Artist
h) Genre
i) Playlist
j) PlaylistTrack

Query:

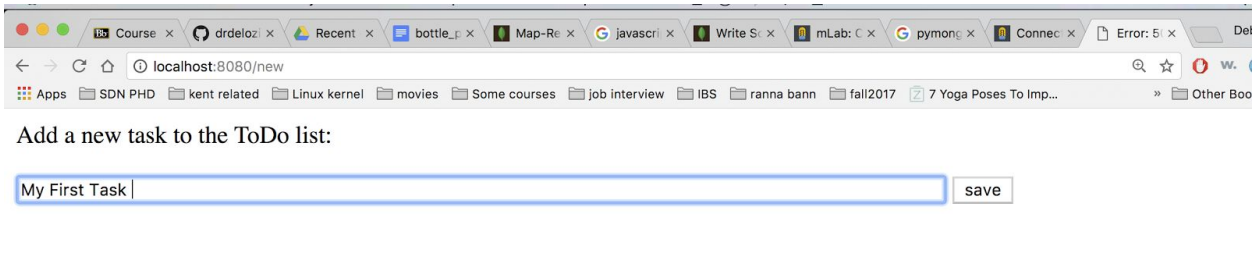
```
SELECT Artist.Name, Genre.Name, COUNT(*)
FROM Playlist, PlaylistTrack,
Track, Album, Artist, Genre
WHERE Playlist.PlaylistId = PlaylistTrack.PlaylistId
AND PlaylistTrack.TrackId = Track.TrackId
AND Track.AlbumId = Album.AlbumId
AND Album.ArtistId = Artist.ArtistId
AND (Genre.Name='Pop' )
AND Genre.GenreID = Track.GenreID
GROUP BY Genre.Name, Artist.Name
```

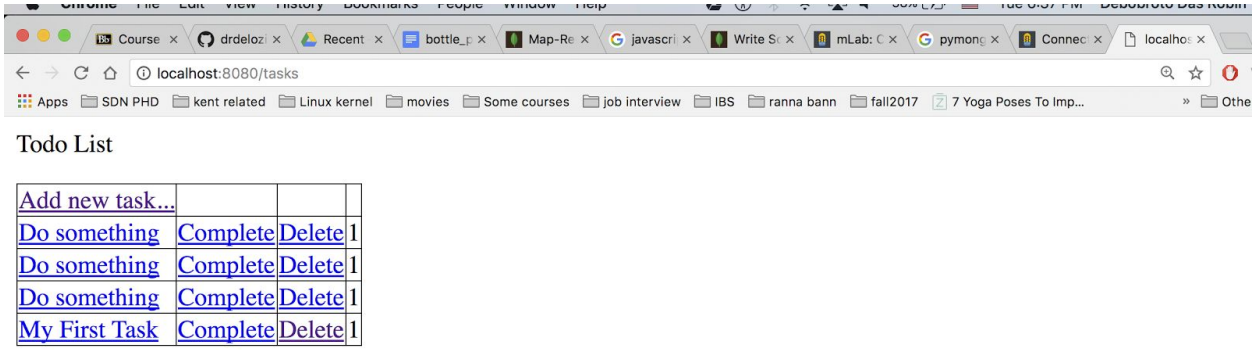
What the query does:

It lists name of all the artists who sings “Pop” songs, and how many times their tracks have been listed in Track table.

Question 3: Web app running with Mongo on MLAB

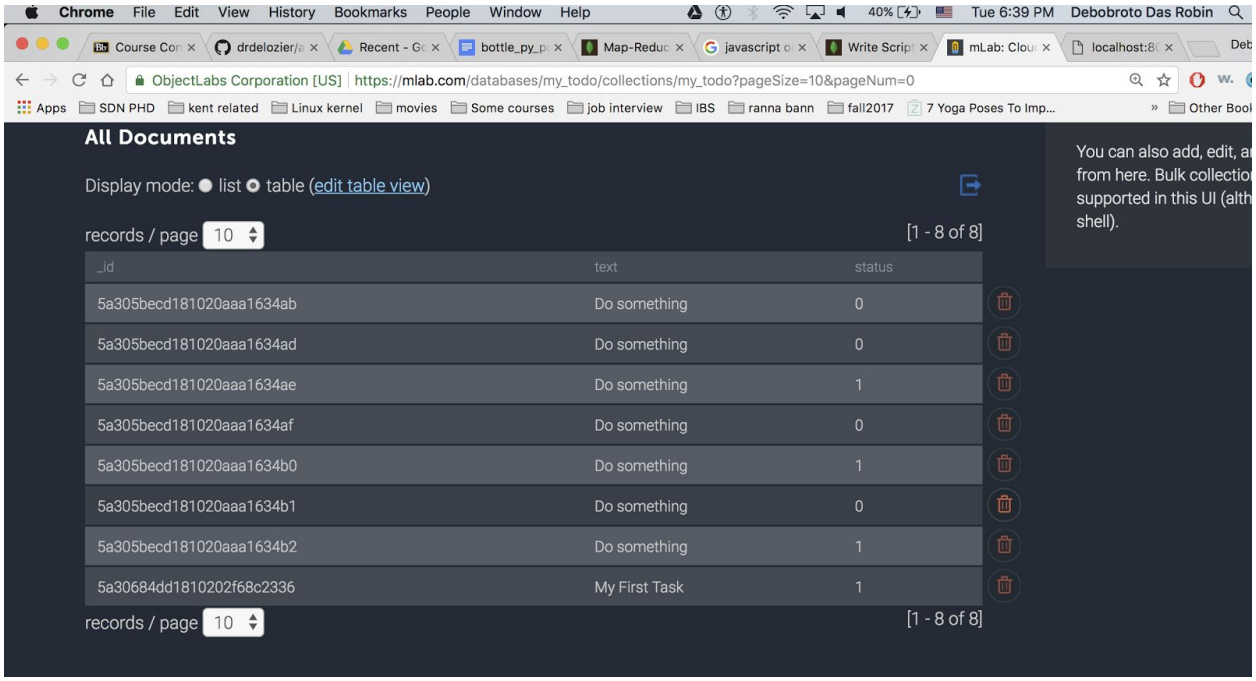
Here are 2 screenshots of my todo app running with mlab mongo instance.





Question 4: Mongo running on server you have set up

I have setup a Mongo instance on mLab. A screenshot is added below:



Here you can see the last entry in the mlab mongo instance's collection . Its task name is "My First Task"

Question 5: A Mongo query aggregation example using the ZIP or Restaurant database for some interesting purpose

Answer:

```
//Gives total population and average population and number of cities of the states
db.zipcodes.aggregate(
  [{ $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } },
  { $group: { _id: "$_id.state", avgCityPop: { $avg: "$pop" }, totalpop: { $sum: "$pop" }, totalCity :{$sum :1 }
} },
            { $sort: { totalPop: 1 } },
             {$out:"total_avg_pop_by_state"}]
 )
```

Question 6: A Mongo map-reduce example using the ZIP or Restaurant database for some interesting purpose

Answer:

What this map-reduce job does: Gives total population and average population and number of cities of the states

Code:

```
var mapFunction = function () {
    var key = this.state;
    var value = {
        total_pop: this.pop,
        city: this.city,
        count :1
    };

    emit(key, value);
};

var reduceFunction = function (key, values) {
    var reducedObject = {
        state: key,
        total_pop: 0,
        count: 0,
        avg_pop : 0
    };

    values.forEach(function (value) {
        reducedObject.total_pop += value.total_pop;
        reducedObject.count += value.count;
    }
    );
    return reducedObject;
};
var finalizeFunction = function (key, reducedValue) {

    if (reducedValue.count > 0)
        reducedValue.avg_pop = reducedValue.total_pop / reducedValue.count;

    return reducedValue;
};
db.zipcodes.mapReduce(mapFunction,
    reduceFunction,
    {
        out: "state_Data",
```

```
        finalize: finalizeFunction
    }
)
```