MLD 2024- Problem set 1

1. Trying out git

This I am assuming was straightforward - the lecture notes should have enough information for this,

2. Create the Stars and Observations tables

In the MLD2024 git repository there is a file ProblemSets/MakeTables/make_tables_python.py that shows how to create these tables using Python.

3. Querying using Python

This is hopefully straightforward

4. Completing the lecture

Where is the FITS image stored for star S5?

```
In [1]: import sqlite3 as lite
In [2]: con = lite.connect("MLD2024.db")
In [3]: rows = con.execute("""Select s.StarID, o.WhereStored
   ...:    From Stars as s Join Observations as o on s.FieldID = o.ID
   ...:    Where s.Star = 'S5' """)
In [4]: for row in rows:
    print("Star {0} is stored at {1}".format(row[0], row[1]))
   ...:
Star 3 is stored at /disks/yaeps-1/StF-045.fits
```

Give me a list of all stars observed with the same FieldID.

There are many ways to answer this. You can do the work in Python or in SQL. Let me focus on the SQL solution - so I will just show the SQL, you can either type this into sqlite3 or do this in Python. There are also different ways to interpret the question. I will also show this in three steps - each step better than the previous. The solution I will go for here is a join with the stars table itself:

First go:

```
select s1.fieldID, s1.Star, s2.Star
from
  Stars as s1
  JOIN Stars as s2
  ON s1.fieldID = s2.fieldID
```

The result of this is (prettifying the output a bit):

```
# FieldID Star Star
    1      S1   S1
    1      S2   S1
    1      S1   S2
    1      S2   S2
    3      S5   S5
    2      S7   S7
```

which is not very satisfying because there is a lot of repetition and symmetric solutions.

If we now consider that we should only print those fields where there is more than one star (interpretation!), we should at least get rid of matches to the star itself.

```
select s1.fieldID, s1.Star, s2.Star
from
  Stars as s1
  JOIN Stars as s2
  ON s1.fieldID = s2.fieldID
Where s1.Star != S2.Star
```

which gives

```
# FieldID Star Star
    1      S2   S1
    1      S1   S2
```

That is better, but still there is a duplication here. The key now is to realise that we have to impose an order instead of a not equal statement. This gives

```
select s1.fieldID, s1.Star, s2.Star
from
  Stars as s1
  JOIN Stars as s2
  ON s1.fieldID = s2.fieldID
Where s1.star > s2.star
```

which then gives:

```
# FieldID Star Star
    1      S2   S1
```

(you can of course swap the order if you wish!)

This is perhaps a type of problem where you would handle it in Python and it would be a lot easier - but if your database is very large that is not a good option.

But this is only one way to view the question - maybe instead you want to see all stars in each field in an organised manner. This is trickier and the best way to handle this is to use python and do:

1. Get a list of all unique fields
2. Loop over these and find all stars in each field.

This is best done in python, here is an example solution

```
import sqlite3 as lite

# Get this from the MakeTables directory if you haven't
# got this already
con = lite.connect('MLD2024.db')

# Find the unique field IDs
rows = con.execute('Select DISTINCT ID From Observations')
fieldIDs = [row[0] for row in rows]

# Loop over the fieldIDs and find those that have
# stars in the Stars table.
for fid in fieldIDs:
    SQLstatement = """
SELECT Star
FROM Stars
WHERE FieldID = "{0}"
""".format(fid)

    rows = con.execute(SQLstatement)

    # This gives us the number of stars in field i
    starIDs = [row[0] for row in rows]
    if len(starIDs)>0:
        print("Field {0} has {1} stars".format(fid, len(starIDs)))
        for sid in starIDs:
            print("    {0}".format(sid))
```

which prints

```
Field 1 has 2 stars
    S1
    S2
Field 2 has 1 stars
    S7
```

```
Field 3 has 1 stars
    S5
```

5. Creating simple tables.

See ProblemSets/1 - SQL and Databases/Solution/make_simple_tables.py for the creation of tables

For the querying there is a second file named query-simple-tables.py that does that. But the code using Pandas is sufficiently simple that we can look at a couple here:

```
con = lite.connect('SimpleTables-default.db')
a)
t = pd.read_sql_query("Select Name, Ra, Decl From MagTable Where B > 16", con)
print(t)
b)
query = """
SELECT m.B, m.R, p.Teff, p.FeH
FROM MagTable as m OUTER LEFT JOIN PhysTable as p
ON m.Name = p.Name
"""
t = pd.read_sql_query(query, con)

c) - almost the same as b but with an extra WHERE FeH > 0
d)
# First get one using a query
col_t = pd.read_sql_query("SELECT Name, B-R as BR FROM MagTable", con)

# Then save it
col_t.to_sql("BRTable", con, if_exists="replace")
```

6. Running SQL queries on the SDSS databases - CasJobs

This should by straightforward.

7. Linear regression

See the `Exploring Regression for problem set - solution.ipynb` in the `ProblemSets/1 - SQL and Databases/Solution` directory.

8. Returning to the SDSS - Finding stars

Log into CasJobs. Here you need to build the results by using the UNION operator:

```
SELECT * into mydb.UnionOfStars2 from
 (
(SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 10 AND psfMag_r < 11)

UNION

 (SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 11 AND psfMag_r < 12)

 UNION

 (SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 12 AND psfMag_r < 13)

   UNION
```

```
 (SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 13 AND psfMag_r < 14)

   UNION

     (SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 14 AND psfMag_r < 15)

   UNION

 (SELECT TOP 100 objID, psfMag_r, psfMag_g-psfMag_r as gr
FROM Star as s
WHERE psfMag_r >= 15 AND psfMag_r < 16)
) as stars
```

9. Aggregating functions in SQL

Lecture notes should be sufficient.

10. the fraction of red galaxies with redshift.

This is most easily achieved by dividing up the query in two: A first that counts the red galaxies grouped by redshift bins, and a second one which counts all galaxies and then outside you combine these two with a JOIN on the redshift.

```
SELECT red.redshift, cast(red.num AS float)/cast(b.num AS float) as fraction
FROM  (
  Select floor(0.5+z*50.)/50 as redshift, count(*) as num
   From SpecPhoto
  Where
   modelMag_u-modelMag_g > 2
  AND
   z between 0.001 and 0.5
  GROUP BY floor(0.5+z*50.)/50
) as red JOIN
(
Select floor(0.5+z*50.)/50 as redshift, count(*) as num
From SpecPhoto
Where
   z between 0.001 and 0.5
GROUP BY floor(0.5+z*50.)/50
) AS b ON b.redshift = red.redshift

ORDER BY red.redshift
```

11. CasJobs in MAST instead

Hopefully straightforward

a) With context set to TESS_v7

```
Select COUNT(GAIA)
FROM catalogRecord
WHERE
  d BETWEEN 0 AND 100
AND
  Teff > 12000
```

which returns 8.

b)

One possible solution:

```
SELECT .2*(.5+floor(Hmag/.2)) as H,
count(*) as num into mydb.HbandHist from CatalogRecord
GROUP BY .2*(.5+floor(Hmag/.2))
ORDER BY H
```