1. [10 pts] Looking at the Schema file (init_dataverse.txt), which dataset/s in AsterixDB can be classified as being in 1NF based on the DDL you’ve been given? The commented section describes what records could be stored in those datasets. For this question only, we consider data stored in datasets with open data types has no extra fields. Write down the dataset name(s) and briefly explain your answer(s).

In 1NF: Interest, Observer, Observation. These datasets only include atomic attributes.

Not in 1NF: Thought, Event, and User datasets are not in 1NF since they contain non-atomic attributes.

2. [10pts] There is another NoSchema file named “noschema_alternative.txt”, which can also be used to create datatypes and datasets. Repeat step 0 but this time use the NoSchema version. You can use the same load dataset as in step 0, but you need to change the ‘USE’ statement to be ‘USE NoSchema’. Like this:

   USE NoSchema;

   LOAD DATASET Observations USING localfs...

Compare the data type definitions in both the Schema and NoSchema. How is the NoSchema different from Schema in terms of number/type of attributes? Is there any difference in the way the data is showing in both versions? (You can make some queries to verify your answer, but you don’t have to attach them to the answer.) Note: the dataverse that contains the schema version is ‘HW8’ and the non-schema version is ‘NoSchema’. You only need to use ‘NoSchema’ for this question.

In the simple version, there are fewer attributes per datatype and each datatype only contains prime attributes. There is no difference in the way the data is showing in both versions and you can query other attributes that are not explicitly defined just as you can the predefined attributes.

3. [10 pts] Looking back at the E-R diagram from the PHLogger conceptual design, compare and contrast the MySQL schema from the past SQL HW assignments with the AsterixDB schema (given in the Schema DDL version). You will find that we have made some different design decisions here in the NoSQL database case. Very briefly explain, after looking at the schema and also exploring the data (e.g., by looking at the DDL statements in the script and by running “SELECT VALUE u FROM Users u LIMIT 10;” and similarly for Events), how we have captured the information from the following E-R entities differently in AsterixDB and what the benefit(s) of our new design probably are:

Users: We collapsed both PHLoggers and Supporters into this dataset and have a utype to distinguish them. Also, we used nested attributes to stored the Contacts and Address information of the PHLogger users. This avoided additional JOIN between multiple tables when fetching the users’ information. Also, we pushed nested a user’s interested group ids as an array attribute, so it can be retrieved efficiently as
well.

*Events*: We include the observation ids that indicate an event rather than having a separate topic dataset.

4. [12 pts] For the user whose id is “97”, print his/her **user’s id, name, email address, most recent heart rate, the kind of the observer that recorded this reading, the start timestamp of this reading.** The recentness of a heart rate reading is determined by its start timestamp. [Result size: 1]

Sample output: {"id": 97, "name": "Gwenn Dooley", "email": "gwenn.dooley@uci.com", "rate": 21, "kind": "watch", "startTimestamp": "2019-04-15T13:00:00.000Z" }

[8 pts] Query:

USE HW8;

SELECT u.id, u.name, u.email, obs.rate, obr.kind, obs.startTimestamp FROM Observations obs, Users u, Observers obr WHERE obs.kind = 'heart_rate' AND obr.ownedBy = u.id AND obs.observerId = obr.id AND u.id = 97 ORDER BY obs.startTimestamp DESC limit 1;

[4 pts] Result:

{ "id": 97, "email": "gwenn.dooley@uci.com", "kind": "glucose monitor", "startTimestamp": "2019-03-21T16:00:00.000Z", "name": "Gwenn Dooley", "rate": 62 }

5. [14 pts] List all the attributes of the top three users who have joined the most number of interest groups (**Hint**: You might want to check out the len() function [here](#)). [Result size: 3]

Sample output: {
"id": 2, "utype": "PHLogger", "contact": [ ], "memberOf": [ "alcoholism 1", "exercise 2", "asthma 1" ], "address": { "state": "NC", "city": "Murazikfort", "street": "Timmy Club", "zipcode": "61183" }, "passwd": "aa8ed233b0aa9b7492bd58910c5f1392", "email": "bethany.macgyver@uci.com", "name": "Bethany MacGyver" }

[7 pts] Query:

USE HW8;

SELECT VALUE u FROM Users u
ORDER BY len(u.memberOf) DESC limit 3;

[4 pts] Result:
Now try the same query on the **NoSchema** version of the PHLogger dataverse (`NoSchema`).

- a) Did it work?
- b) Were the results different?
- c) What does this tell you?

It works and there is no change to the above SQL++ query. You can query for non-defined attribute the same way you would the pre-defined ones.

6. [14 pts] List the **ids** of the users who are members of groups with the topic of “diabetes” and have an average heart rate greater than 85.  
(HINT: You might find ‘IN’ useful for checking the existence of a nested array element.) [Result size: 4]

Sample output: `{ "id": 9 }...`

[10 pts] Query:

```
USE HW8;
SELECT u.id
FROM Observations obs, Users u, Observers obr, Interests i
WHERE obs.kind = 'heart_rate' AND obr.ownedBy = u.id AND obs.observerId = obr.id AND i.name IN u.memberOf AND i.topic = "diabetes"
GROUP BY u.id
HAVING avg(obs.rate) > 85;
```

[4 pts] Result:

```
{ "id": 9 }
{ "id": 21 }
{ "id": 25 }
{ "id": 32 }
```
7. [18 pts] Write a query to return the attributes of the user with id "85", together with the texts of all thoughts posted by this user, all blood pressure readings (including both diastolic and systolic) and heart rate readings recorded by observers owned by this user. (You can use user.* to return all attributes from a user. You might also want to read the SELECT VALUE info here and review the SQL++ tutorial info on how to form a nested array of values.) [Result size: 1]

Sample output: 
```
{
  "thought_texts": ["Today I went to an AA meeting."],
  "blood_pressure": [ {
    "diastolic": 93, "systolic": 128
  }, {
    "diastolic": 103, "systolic": 171
  } ],
  "heart_rates": [ 70, 94 ],
  "id": 11,
  "utype": "PHLogger",
  "contact": [],
  "memberOf": [ "HIV 2" ],
  "address": {
    "state": "SC",
    "city": "New Dovie",
    "street": "Lind Roads",
    "zipcode": "36502"
  },
  "passwd": "b643b952877ba9468e0a40734626f8e7",
  "email": "jewell.greenfelder@uci.com",
  "name": "Jewell Greenfelder"
}
```

To help you understand, here is a formatted version of the answer:

```
Results:
```
```
-
  - thought_texts: [
      "Today I went to an AA meeting."
  ],
  - blood_pressure: [
      {
        "diastolic": 103,
        "systolic": 171
      },
      {
        "diastolic": 93,
        "systolic": 128
      }
  ],
  - heart_rates: [ 70, 94 ],
  - id: 11,
  - utype: "PHLogger",
  - contact: [],
  - memberOf: [ "HIV 2" ],
  - address: {
      "state": "SC",
      "city": "New Dovie",
      "street": "Lind Roads",
      "zipcode": "36502"
    },
  - passwd: "b643b952877ba9468e0a40734626f8e7",
  - email: "jewell.greenfelder@uci.com",
  - name: "Jewell Greenfelder"
```
```
Query:
USE HW8;
FROM Users u
WITH
    thought_texts AS (SELECT value t.text FROM Thoughts t WHERE t.phlid = u.id),
    heart_rates AS (SELECT VALUE obs.rate FROM Observations obs, Observers obr
        WHERE obs.kind = "heart_rate" AND obs.observerId = obr.id AND obr.ownedBy = u.id),
    blood_pressure AS (SELECT obs.diastolic, obs.systolic FROM Observations obs, Observers obr
        WHERE obs.kind = "blood_pressure" AND obs.observerId = obr.id AND obr.ownedBy = u.id)
WHERE u.id = 85
SELECT u.*, thought_texts, blood_pressure, heart_rates;

Result:
{ "thought_texts": [ "It was easy for me to breathe today.", "I resisted the urge to eat ice-cream today.", "I relapsed and I do not feel that bad." ],
  "blood_pressure": [ { "diastolic": 63, "systolic": 123 }, { "diastolic": 107, "systolic": 103 }, { "diastolic": 138, "systolic": 190 } ],
  "heart_rates": [ 69, 96, 86 ],
  "id": 85,
  "utype": "PHLogger",
  "contact": [ "640-577-0013" ],
  "memberOf": [ "alcoholism 3" ],
  "address": { "state": "CT", "city": "East Elishamouth", "street": "Elbert Fields", "zipcode": "98199" },
  "passwd": "8fd4f4773894d0316332121d0fa60c8c",
  "email": "ammie.gleason@uci.com",
  "name": "Ammie Gleason"
}

As NoSQL databases provide nested attributes, we could have pushed all the Thoughts, Observations into the Users dataset. Briefly explain what could be the reason(s) that we didn’t do that.
If we store the thoughts and observations into a user’s record, the user’s record will get large when there are more thoughts and observations in an ever-growing fashion. Also, inserting a new thought or observation would need to access the whole user’s record. This can be expensive, especially when this record is large.
8. [10 pts] Write a query to print the user id and the number of thoughts of users who have posted at least 9 thoughts. [Result size: 6]
Sample output: { "id": 1, "cnt": 1 }.. 

[7 pts] Query:
USE HW8;
SELECT u.id AS id, count(*) as cnt
FROM Users u, Thoughts t
WHERE u.id = t.phlid
GROUP BY u.id
HAVING count(*) >=9;

[3 pts] Result:
{ "id": 37, "cnt": 9 }
{ "id": 49, "cnt": 9 }
{ "id": 79, "cnt": 9 }
{ "id": 92, "cnt": 9 }
{ "id": 31, "cnt": 9 }
{ "id": 78, "cnt": 9 }