Last Name: Yangkaiqi First Name: Ren Student ID: 93672321

1. [10 pts] Take a look at the load script you ran in the instructions (`swoosh_schema.sqlpp`) Which of the dataset(s) in AsterixDB can be classified as ***not*** being in 1NF based on the DDL that you've been given? For this question only, we will consider the schema'ed data, i.e., the data stored in the AsterixDB dataverse and datasets that have predeclared ADM data types. Identify and list the 1NF and non-1NF datasets' name(s) and briefly explain your answer(s). (*Reminder*: 1NF = Flat atomic-valued relational data.)

In 1NF: Attended, Course, EnrolledIn, Meeting, Recording, Recurrence, Teaches, ThumbsUp
They all only have atomic attributes

Not 1NF: Post (Has multivalued attributes: topics)

Users(Has multivalued attributes: names-first name last name)

Watched(Has multivalued attributes: watched from watched to)

2. [10pts] Repeat **step 0** (the setup/loading step) from the instructions document, but this time use the **schemaless** version of SWOOSH. The schemaless version is the subject of the load script named `swoosh_schemaless.sqlpp`. The data files are the same ones as in step 0, but you will need to change the 'USE' statement at the top of the file to say `USE swoosh_schemaless;`. Examine and compare the DDL of both schema versions. How is the DDL for the schemaless version different from the DDL for the schema version in terms of its number and types of attributes? Is there any difference in the way the data appears in the two versions? (**Note**: Again, the dataverse that contains the schema version is named `swoosh_schema` while the schemaless one is named `swoosh_schemaless`.)

In the NoSchema version, there is only one attribute per data type (and it is not a part of the original schema). 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 SWOOSH conceptual design, compare and contrast the relational (PostgreSQL) version of the schema from the past SQL HW assignments with the AsterixDB schema here (the one given in the schema DDL version). You will find that we have made some rather different design decisions here in the NoSQL database case. Very briefly explain, after looking at the AsterixDB schema and after also exploring the data (e.g., by looking at the DDL statements in the script and running exploratory queries like "`SELECT VALUE u FROM Users u LIMIT 20;`") for **Users** and similarly for **Watched**, how we have captured the information from each of the following E-R entities differently in AsterixDB and what the benefit(s) (hint: JOIN operations) of this new design probably are.

User: Instructors and students this identification are folded into the User dataset. There are two flags that indicate whether the user is an instructor or a student. This helps us to save a join when querying for instructors and students separately. We also include education attributes for users rather than have a separate education dataset.

Watched*:* Watchedsegmebt is folded into watched, which saved join when a query for watched start time and end time.

4. [7 pts] *Use the dataverse* `swoosh_schema` *for problems 4 and beyond.* For users that are both instructors and students, list their user ids, their full name (in two **separate** fields, first name and last name), and their occupation. Users **who do not have an occupation should not be included** in your result. [Result size: 9]

Sample output:

```
{ "user_id": "142", "first_name": "Patricia", "last_name": "Moreno", "occupation":
"Statistician" }
```

[4 pts] Query:

USE swoosh_schema;

Select U.user_id, U.name.first_name, U.name.last_name, U.occupation

From Users U

Where U.isstudent = True and U.isinstructor = True and U.occupation != ""

[3 pts] Result:

```
{"user_id":"142","first_name":"Patricia","last_name":"Moreno","occupation":"Statistician"}
{"user_id":"175","first_name":"Diana","last_name":"Silva","occupation":"Environmental manager"}
{"user_id":"179","first_name":"Michael","last_name":"Stanley","occupation":"Cytogeneticist"}
{"user_id":"22","first_name":"Charles","last_name":"Shields","occupation":"Systems analyst"}
{"user_id":"235","first_name":"Kimberly","last_name":"Nichols","occupation":"Lecturer, higher education"}
{"user_id":"333","first_name":"Mr.","last_name":"Charles","occupation":"Accountant, chartered public finance"}
{"user_id":"410","first_name":"Barbara","last_name":"Smith","occupation":"Chief Executive Officer"}
{"user_id":"60","first_name":"Wesley","last_name":"Harris","occupation":"Advertising account planner"}
{"user_id":"73","first_name":"Randall","last_name":"Ponce","occupation":"Translator"}
```

5. [13 pts] For students whose first name is 'Emily' , write a query to get the students' user_id, full name (in two **separate** fields), email, and a field containing a list of names of the courses they are enrolled in. [Result size: 2]

Sample output:

```
{ "user_id": "17", "first_name": "Emily", "last_name": "Arroyo", "email":
"deborahbryant@mit.edu", "courseNames": [ "Acting", "Communications 102", "Calculus
IV" ] }
```

[7 pts] Query:

```
USE swoosh_schema;
SELECT U.user_id,U.name.first_name, U.name.last_name,U.email,
(Select Value C.course_name
From Course C, EnrolledIn E
Where U.user_id = E.user_id and C.course_id = E.course_id) AS courseNames
From Users U
Where U.name.first_name = "Emily" and U.isstudent and U.isinstructor = False
```

[3 pts] Result:

```
{"user_id":"17","first_name":"Emily","last_name":"Arroyo","email":"deborahbryant@mit.edu","courseNames":["Acting","Communications 102","Calculus IV"]}
{"user_id":"412","first_name":"Emily","last_name":"Carlson","email":"jeremy39@mit.edu","courseNames":["Anatomy II","Art History","World History II","Calculus II"]}
```

[3 pts] Now try the same query on the **schemaless** version of the SWOOSH dataverse (**'swoosh_schemaless'**). Did it work? Were the results different? What does this tell you about querying typed versus untyped data in SQL++?

It works and you can query for non-defined attributes the same way you would the pre-defined ones.

6. [10 pts] Write a query to print the user_ids of instructors and a list of their corresponding majors for instructors where the number of **distinct** majors that they have majored in is equal to 5. Order your result in descending order (on the user_id) and limit the number of results to the top five (don't worry here about the order not being correct numerically due to user_id being a string/text). [Result size: 5]

Hints:

- See the documentation (and lecture notes) for array_count().
- Check out the LET-clause in AsterixDB's documentation, as it may help simplify your query. You do not need to repeat yourself with SQL++ :-)
- Remember that the result of a subquery in SELECT is always an array! (Life gets easier once you remove the straight-jacket of 1NF :-))

Sample output:

```
{"user_id": "131", "major_list": ["Biology","Financial Engineering","Mechanical
Engineering", "Music", "Physics"]}
```

[7 pts] Query:

```
USE swoosh_schema;
SELECT U.user_id, major_list
FROM Users U
LET major_list = (SELECT DISTINCT VALUE UE.major FROM U.education UE)
WHERE array_count(major_list) = 5 AND U.isinstructor
ORDER BY U.user_id DESC
LIMIT 5;
```

[2 pts] Is your query result in 1NF? Why or why not?

`NO, because there are multivalues in majors`

[1 pts] Result:

```
{"user_id":"94","major_list":["Business Management","Chemistry","English","Mechanical
Engineering","Nursing"]}
{"user_id":"9","major_list":["Electrical Engineering","Financial Engineering","International
Studies","Music","Physics"]}
{"user_id":"55","major_list":["Chemistry","International Studies","Mathematics","Mechanical
Engineering","Performing Arts"]}
{"user_id":"497","major_list":["Aerospace Engineering","Electrical Engineering","Financial
Engineering","History","Mechanical Engineering"]}
```

```
{"user_id":"479","major_list":["Aerospace Engineering","History","Mathematics","Nursing","Pre-Medicine"]}
```

7. [10 pts] Write a query that - for posts that have been posted before "2020-08-10 00:00:00", that contain the topic "project2", and whose posting users have liked more than 4 posts - prints the post_ids, their poster' user_ids, and a list of the post_ids that the poster liked. [Result size: 2]

Hints:

- Check out the lecture and docs about the existential SOME clause. You might find it useful. :-)
- You might need to cast the date to datetime in order to compare it, i.e. DATETIME("2020-08-10 00:00:00")

Sample output:

```
{"post_id": "710","user_id": "234","liked_posts":
["109","230","44","591","64","92"]}
```

[7 pts] Query:

```
USE swoosh_schema;
SELECT p.post_id, p.user_id, liked_posts
FROM Post p
LET liked_posts = (SELECT VALUE tu.post_id FROM ThumbsUp tu WHERE p.user_id = tu.user_id)
WHERE (some tp in p.topics SATISFIES tp = "project2") AND array_count(liked_posts) > 4 AND p.created_at < DATETIME("2020-08-10T00:00:00")
;
```

[3 pts] Result:

```
{"post_id":"710","user_id":"234","liked_posts":["109","230","44","591","64","92"]}
{"post_id":"984","user_id":"52","liked_posts":["428","445","79","918","984"]}
```

8. [10 pts] Write a query that analyzes instructors' alma maters' popularity. It should print all of the ***schools*** that one or more instructors have attended and the number of distinct instructors who have attended them. Order your result in descending popularity order and limit the number of results to the top five. [Result size: 5]

Hints:

- Check the UNNEST clause here and/or the shorthand for UNNEST as a join clause here.

Sample output:

```
{"university": "University of Maryland", "instructor_cnt": 23}
```

[7 pts] Query:

USE swoosh_schema;

SELECT UE.school AS schools, COUNT(DISTINCT U.user_id) AS instructor_cnt FROM Users U, U.education AS UE

WHERE U.isinstructor = true

GROUP BY UE.school

ORDER BY instructor_cnt DESC

LIMIT 5;

[2 pts] Result:

```
{"schools":"University of Maryland","instructor_cnt":23}
{"schools":"Virginia Tech","instructor_cnt":23}
{"schools":"UCR","instructor_cnt":23}
{"schools":"USD","instructor_cnt":21}
{"schools":"MIT","instructor_cnt":21}
```

[1 pts] Is the result in 1NF? Why?

It is !NF only have atomic attributes

9. [10 pts] We would like to perform analytics on Posts, specifically through aggregations of post types and their associated meetings. Use the **ROLLUP** operator to show the total number of post authors for each (post_type, meeting_id) combination as well as for each post_type and the overall number as well. Order your results on the combination (post_type, meeting_id) in ascending order, and limit your results to 5. [Result size: 5]

Sample output:

```
{"post_type": "note","meeting_id": null,"author_cnt": 511}
```

[8 pts] Query:

USE swoosh_schema;

SELECT post_type, meeting_id, COUNT(P.user_id) AS author_cnt FROM Post P

GROUP BY ROLLUP(post_type, meeting_id)

ORDER BY (post_type, meeting_id) ASC

[2 pts] Result:

```
{"schools":"University of Maryland","instructor_cnt":23}
{"schools":"Virginia Tech","instructor_cnt":23}
{"schools":"UCR","instructor_cnt":23}
{"schools":"USD","instructor_cnt":21}
{"schools":"MIT","instructor_cnt":21}
```

10. [10 pts] An additional analytical query has come to mind: For each course, rank the courses in descending order of their instructor count (how many instructors have taught the course). Use SQL++'s window-based aggregation (i.e., the **OVER** clause) to find the ranks. Print out the course IDs, course names, instructor counts, and ranks. Limit your results to the top 3 **ranks**. [Result size: 3]
Note: do not use LIMIT here!

Sample output:

```
{"course_id": "23","course_name": "Informatics","inst_count": 28,"rank": 1}
```

[8 pts] Query:
USE swoosh_schema;
SELECT Result.course_id, Result.course_name, Result.inst_count, Result.rank FROM
(SELECT C.course_id, C.course_name, inst_count, rank() OVER (ORDER BY inst_count DESC) AS rank
FROM Course C
LET inst_count = array_count((SELECT VALUE T.user_id
FROM Teaches T
WHERE C.course_id = T.course_id)) ORDER BY rank ASC) AS Result
WHERE Result.rank <= 3;

[2 pts] Result:
{"course_id":"23","course_name":"Informatics","inst_count":28,"rank":1}
{"course_id":"0","course_name":"Communications 101","inst_count":27,"rank":2}
{"course_id":"46","course_name":"Communications 102","inst_count":27,"rank":2}