Homework 8: NoSQL (100 points)

1. [10 pts] Looking at the Schema file (Dataset_Schema_Init.txt), which dataset/s in AsterixDB can be classified as being in 1NF based on the DDL you’ve been given? Write down the dataset name(s) and briefly explain your answer(s).

In 1NF: Course, Class, Assignment, Likes, Does, Takes, Offers. These datasets only include atomic attributes.

User and Post datasets are not in 1NF since they contain non-atomic attributes.

2. [10pts] Repeat step 0 but this time use the NoSchema version of PEEZa.com. This is the script named ‘Dataset_NoSchema_Init.txt’. The data file is the same one as in step 0 but you need to change the ‘USE’ statement at the top of the file to say ‘USE cs122aNoSchema;’. Compare the DDL of both schema versions. How is the DDL for NoSchema different from the DDL for Schema version in terms of number/type of attributes? Is there any difference in the way the data is showing in both versions?

Note: the dataverse that contains the schema version is ‘cs122aSchema’ and the non-schema version is ‘cs122aNoSchema’.

In the NoSchema 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 PEEZa 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 User u LIMIT 20;” and similarly for Post), 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:

**User:** We include both Student and Instructor entities in the User dataset and have a user_kind attribute to indicate the kind of user. In this design, we can save a join when querying for students and instructors separately. We include the nested/repeating phone attribute for each user as one of its attributes rather than having a separate phone dataset.

**Post:** We include the nested/repeating topic information for each post as one of its attributes rather than having a separate topic dataset.
4. [10 pts] Write a query to print the ids of users, the corresponding post_id and score from popular posts (defined as having a current popularity score greater than 10) about the assignment with id "112". [Result size: 2]

Sample output: {"user_id": "1128", "post_id": "1828", "popularity": 18 }

[7 pts] Query:
SELECT p.user_id, p.post_id, p.popularity FROM Post p WHERE p.assign_id = "112" AND p.popularity > 10;

[3 pts] Result:
{ "user_id": "1128", "post_id": "1828", "popularity": 18 }
{ "user_id": "625", "post_id": "1822", "popularity": 74 }

5. [10 pts] For those students majoring in "ART" who have taken the course CS222 (i.e., with cno "222" and dept "CS") with a grade no less than 98, list their user information. [Result size: 3]

Sample output: { "user_id": "628", "user_kind": [ "Student" ], "name": { "first_name": "Marley", "last_name": "Spinka" }, "email": "marley.spinka@yahoo.com", "phone": [ { "phoneType": "Office", "number": "(008) 086-8419" }, { "phoneType": "Office", "number": "918.135.2431" } ], "major": "ART" }

[4 pts] Query:
SELECT VALUE u FROM User u, Takes t
WHERE u.major = "ART"
AND t.user_id = u.user_id
AND t.dept = "CS" AND t.cno = "222"
AND t.grade>=98;

[3 pts] Result:
{ "user_id": "1348", "user_kind": [ "Student" ], "name": { "first_name": "Delpha", "last_name": "Denesik" }, "email": "delpha.denesik@hotmail.com", "phone": [ { "phoneType": "Mobile", "number": "(917) 858-3805" }, { "phoneType": "Office", "number": "(939) 939-8063" }, { "phoneType": "Home", "number": "(061) 656-1000" } ], "major": "ART" }

{ "user_id": "1527", "user_kind": [ "Student" ], "name": { "first_name": "Lorenzo", "last_name": "Lubowitz" }, "email": "lorenzo.lubowitz@yahoo.com", "phone": [ { "phoneType": "Office", "number": "(053) 414-2013" }, { "phoneType": "Home", "number": "1-470-791-1053" }, { "phoneType": "Home", "number": "767.648.5060" } ], "major": "ART" }

{ "user_id": "628", "user_kind": [ "Student" ], "name": { "first_name": "Marley", "last_name": "Spinka" },"
3 pts] Now try the same query on the NoSchema version of the PEEZa dataverse ('cs122aNoSchema'). Did it work? Were the results different? 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. [10 pts] Write a query to find all instructors who have taken the course CS122A (cno "122A" dept "CS") with a grade of at least 95 and list his/her user_id and the actual grade received. (HINT: You might find 'IN' useful for checking the existence of a nested array element.) [Result size: 4]
Sample output: { "user_id": "17", "grade": 99.0 }

[7 pts] Query:
```sql
SELECT u.user_id, t.grade
FROM User u, Takes t
WHERE u.user_id = t.user_id
AND "Instructor" IN u.user_kind
AND t.dept = "CS"
AND t.cno = "122A"
AND t.grade >= 95;
```

[3 pts] Result:
```
{ "user_id": "1501", "grade": 99.0 }
{ "user_id": "1681", "grade": 98.0 }
{ "user_id": "17", "grade": 99.0 }
{ "user_id": "291", "grade": 97.0 }
```

7. [10 pts] Write a query to print the course title, level and list of associated class numbers for those Classes where the class has a maximum number of students of more than 150 and the course title contains "Data" as a word (not just as a substring) (e.g., so "Database" does not count). (You may follow this doc link to learn about full-text search queries, and you might also want to read the SELECT VALUE info here and review the SQL++ tutorial's info on how to form a nested array of values.) [Result size: 3]
Sample output: {"title": "Principles of Data Management", "level": "Higher Division", "class_nums": ["266", "342", "21"]}

[7 pts] Query:
```sql
SELECT c.title, c.level,
       (SELECT VALUE s.class_no FROM Class s
```
WHERE s.dept = c.dept
AND s.cno = c.cno
AND s.max_students > 150
) as class_nums
FROM Course c
WHERE ftcontains(c.title, "Data");

[3 pts] Result:
{"title": "Introduction To Data Management", "level": "Lower Division", "class_nums": [ "188", "257", "179" ]}
{"title": "Principles of Data Management", "level": "Higher Division", "class_nums": ["266", "342", "21"] }
{"title": "Principles of Data Management", "level": "Graduate", "class_nums": [ "3", "344", "106", "346", "55"]}

8. [10 pts] Write a query to print the user_id and the number of posts of users who have posted at least 225 times. [Result size: 4]
Sample output: { "user_id": "1086", "cnt": 230 }

[4 pts] Query:
SELECT p.user_id, count(*) AS cnt FROM Post p
GROUP BY p.user_id
HAVING count(*) >= 225;

[3 pts] Result:
{ "user_id": "1086", "cnt": 230 }
{ "user_id": "1115", "cnt": 228 }
{ "user_id": "1499", "cnt": 229 }
{ "user_id": "736", "cnt": 229 }

[3 pts] Now try the same query on the NoSchema version of the PEEZa dataverse ('cs122aNoSchema'). Did it work? Were the results different? 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.

9. [10 pts] Write a query to print the last_name, user_id, and phone list for those users who have more than 5 phones and where at least one of the phones is an "Office" phone. (Hint: You might want to check out the len() function here, and you may find the existential SOME clause useful) [Result size: 2]
Sample output: {
  "user_id": "1455", "last_name": "Huel", "phone": [
    { "phoneType": "Mobile", "number": "990.349.2557" },
    { "phoneType": "Office", "number": "(004) 631-6290" },
    { "phoneType": "Office", "number": "1-593-582-6239" },
    { "phoneType": "Office", "number": "660-368-0345" },
    { "phoneType": "Home", "number": "587.835.0650" },
    { "phoneType": "Home", "number": "984.595.0848" }
  ]
}

[7 pts] Query:
SELECT u.user_id, u.name.last_name, u.phone
FROM User u
WHERE len(u.phone) >5
AND (SOME p IN u.phone SATISFIES p.phoneType = "Office" ) ;

Alternative:
SELECT u.user_id, u.name.last_name, u.phone
FROM User u
WHERE SOME p IN u.phone SATISFIES p.phoneType = "Office" AND len(u.phone) >5 ;

[3 pts] Result:
{
  "user_id": "1068", "last_name": "Keebler", "phone": [
    { "phoneType": "Mobile", "number": "1-094-822-8754" },
    { "phoneType": "Mobile", "number": "1-733-750-3547" },
    { "phoneType": "Mobile", "number": "733-000-7438" },
    { "phoneType": "Office", "number": "(106) 956-6520" },
    { "phoneType": "Office", "number": "1-140-107-6973" },
    { "phoneType": "Home", "number": "(842) 205-0447" },
    { "phoneType": "Home", "number": "683-855-0851" }
  ]
}

{
  "user_id": "1455", "last_name": "Huel", "phone": [
    { "phoneType": "Mobile", "number": "990.349.2557" },
    { "phoneType": "Office", "number": "(004) 631-6290" },
    { "phoneType": "Office", "number": "1-593-582-6239" },
    { "phoneType": "Office", "number": "660-368-0345" },
    { "phoneType": "Home", "number": "587.835.0650" },
    { "phoneType": "Home", "number": "984.595.0848" }
  ]
}

10. [10 pts] Write a query that prints the user_id of those users whose title is "Professor" and who do not own any phone along with the post_id, post topic(s), and popularity score of their most popular post(s). (You may need to see the "unknown value comparison" part of this doc link). [Result size: 8]
Sample output: {
  "user_id": "12", "post_id": "8788", "popularity": 98, "topic": [ "Logistics" ]
}

[7 pts] Query:
SELECT u.user_id, p.post_id, p.topic, p.popularity
FROM User u, (SELECT MAX(p.popularity) AS maxScore, p.user_id
                 FROM Post p GROUP BY p.user_id
             ) AS maxPost,
Post p
WHERE u.user_id = maxPost.user_id
AND p.user_id = maxPost.user_id
AND p.popularity = maxPost.maxScore
AND u.phone IS MISSING
AND u.title = "Professor";

[3 pts] Result:
{ "user_id": "475", "post_id": "3165", "popularity": 93, "topic": [ "Homework", "Lectures" ] }
{ "user_id": "518", "post_id": "6831", "popularity": 99 }
{ "user_id": "842", "post_id": "10866", "popularity": 99, "topic": [ "Logistics" ] }
{ "user_id": "12", "post_id": "8788", "popularity": 98, "topic": [ "Logistics" ] }
{ "user_id": "123", "post_id": "10490", "popularity": 99, "topic": [ "Lectures", "Logistics" ] }
{ "user_id": "1499", "post_id": "11467", "popularity": 99, "topic": [ "Lectures" ] }
{ "user_id": "1714", "post_id": "7838", "popularity": 89, "topic": [ "Logistics" ] }
{ "user_id": "649", "post_id": "1531", "popularity": 99, "topic": [ "Exams", "Other" ] }

[EXTRA CREDIT]

11. [10 pts] Come up with an interesting/creative analytical question of your own about this collection of data -- something that was not asked in the previous SQL-based assignments, that exercises the SQL++ GROUP BY functionality described in the SQL++ tutorial, and that involves something “non-flat” in the result or query predicate -- and then write an AsterixDB query to answer it. Show both the English and SQL++ versions of your query as well as the results that you obtain. [Result size: Up to you!]

Query Explanation:

Query:

Result: