Homework 8: NoSQL (100 points)

Due Date: Friday, Jun 8 (6:00 PM)

1. [10 pts] Which dataset/s in AsterixDB can still 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: Reports, Recommendation, Has_read, Refers_to.
These datasets only include atomic attributes.

Additional note: User, Publisher, and Article datasets are not in 1NF since they contain non-atomic attributes.

2. [10 pts] Looking back at the E-R diagram from the conceptual design, compare and contrast the MySQL schema and the AsterixDB schema (given in the DDL). You will find that we have made some different design decisions here in the NoSQL database case. Briefly explain, after looking at the schema and data, 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 the information in Detector and Poster into the User dataset. In this design, we can save one join when querying for posters and detectors. Also, we don’t expect this information to be modified frequently and it won’t grow to be too big. (If the poster and detector information could grow into big lists, that could lead to big user entities which would not be efficient for user queries. For example, that’s why we don’t make Article an attribute of users who are posters.)

Article: We include the topics information into the Article dataset rather than having a separate topic dataset.
3. [10 pts] Write a query to print the userid of posters who have recommended at least one article that they posted. Each userid should be printed only once. [Result size: 3]

Sample output: {"userid": 3 }

[7 pts] Query:
USE cs122a;
SELECT DISTINCT r.from_userid AS userid
FROM Recommendation r
WHERE r.from_userid = r.article_userid;

[3 pts] Result:
{ "userid": 3 }
{ "userid": 6 }
{ "userid": 1 }

4. Write a query to print the title of articles that were written by posters who have posted more than one article. (Check the DDL of the dataset to determine the attribute names.) [Result size: 4]

Sample output: { "title": "White House Hosts Easter" }

[7 pts] Query:
USE cs122a;
SELECT aa.title
FROM Article aa, (SELECT a.userid
                    FROM Article a
                    GROUP BY a.userid
                    HAVING COUNT(a.userid) > 1
                ) AS bb
WHERE aa.userid = bb.userid;

[3 pts] Result:
{ "title": "Inauguration of President Trump" }
{ "title": "iPhone 6 is released" }
{ "title": "White House Hosts Easter" }
{ "title": "iPhone 8 to be announced in September" }
5. [10 pts] Write a query to print the userid of users who became a poster after 2016/01/01 00:00:00 and currently work for a publisher. (You may need to see the “unknown value comparison” part of this doc link.) [Result size: 2]
Sample output: { “userid”: 1}

[7 pts] Query:
USE cs122a;
SELECT u.userid
FROM User u
WHERE u.posterInfo.poster_since > DATETIME("2016-01-01T00:00:00Z")
    AND u.posterInfo.publisherid IS NOT MISSING;

[3 pts] Result:
{ "userid": 1 }
{ "userid": 4 }

Note: AsterixDB, by providing open data-typing, gives developers more flexibility when working with different data formats. In our TrueOnlyUSNews dataset, we snuck one type into the User dataset so that you can experience this feature first-hand. The posterInfo attribute is not defined in UserType, yet the records have that attribute and they can be inserted smoothly into the User dataset. Also, you will see that you can query this attribute just as you can the predefined attributes. To help you understand what to expect, here’s what the full definition of the attribute’s type would be if it had actually been predefined:

PosterInfoType {
    poster_since: datetime,
    publisherid: int?,
    works_since: datetime?
}
6. [10 pts] When importing the data from MySQL, you have realized that some of user data was not imported correctly due to a human mistake. In one instance, even though a user record contains an instance of PosterInfoType, its class attribute is set to “detector”. Write a query that prints the userid of such posters who were accidently classified as detectors. [Result size: 1]
Sample output: { "userid": 23 }

[7 pts] Query:
USE cs122a;
SELECT u.userid FROM User u
WHERE u.class="detector" AND u.posterInfo.poster_since IS NOT MISSING;

[3 pts] Result:
{ "userid": 23 }

7. [10 pts] Write a query to print the userid and degree (level, year, major, and school) information of users who graduated from one of the “University of California” campuses. (You may need to see the “unnest” clause part of this doc link.) [Result size: 2]
Sample output: { "userid": 4, "level": "Bachelors", "year": 1995, "major": "Journalism", "school": "University of California, Irvine" } { "userid": 5, "level": "Bachelors", "year": 2013, "major": "Sport Studies", "school": "University of California, Los Angeles" }

[7 pts] Query:
USE cs122a;
SELECT u.userid , e.level, e.year, e.major, e.school
FROM User u
UNNEST u.degree e
WHERE e.school LIKE 'University of California%';

Additional note: the following WHERE will also work.
... WHERE ftcontains(e.school, {"University","of","California"});
... WHERE ftcontains(e.school, ["University","of","California"]);

[3 pts] Result:
{ "userid": 4, "level": "Bachelors", "year": 1995, "major": "Journalism", "school": "University of California, Irvine" }
{ "userid": 5, "level": "Bachelors", "year": 2013, "major": "Sport Studies", "school": "University of California, Los Angeles" }
8. [10 pts] Write a query that, for each detector, prints the reason and set of topics of the article that have been reported by that detector. The reason for each report should include the keyword “incorrect” (case insensitive). (You may want to look back at “Query 3: Nested Outer Join” in the SQL++ Primer for reference and at this doc link to learn about keyword queries.) [Result size: 6]
Sample output: { "reason": "Incorrect Team", "topic": [ [ "NBA", "Sports" ] ] }

[7 pts] Query:
USE cs122a;
SELECT r.reason, (SELECT VALUE a.topic
    FROM Article a
    WHERE r.article_userid = a.userid AND r.article_articleid = a.articleid) AS topic
FROM Reports r WHERE ftcontains(r.reason, "incorrect");

[3 pts] Result:
{ "reason": "Incorrect Team", "topic": [ [ "NBA", "Sports" ] ] }
{ "reason": "Incorrect Team", "topic": [ [ "NBA", "Sports" ] ] }

Inner join case (also OK):

Query:
USE cs122a;
SELECT r.reason, a.topic
FROM Reports r, Article a WHERE ftcontains(r.reason, "incorrect") AND r.article_userid = a.userid AND r.article_articleid = a.articleid;

Result:
{ "reason": "Kinda incorrect", "topic": [ "Apple", "Phones", "Technology" ] }
{ "reason": "Incorrect year", "topic": [ "Globe", "NASA", "NOAA", "Weather" ] }
{ "reason": "Incorrect year", "topic": [ "Globe", "NASA", "NOAA", "Weather" ] }
{ "reason": "Kinda incorrect", "topic": [ "Apple", "Phones", "Technology" ] }
{ "reason": "Incorrect Team", "topic": [ "NBA", "Sports" ] }
{ "reason": "Incorrect Team", "topic": [ "NBA", "Sports" ] }
9. [10 pts] Write a query that prints the title, quality and popularity of those articles that have received an individual “like score” of 3 or 4 at least once. Each article should be printed only once. [Result size: 4]
Sample output: { "title": "Inauguration of President Trump", "quality": "Clean", "popularity": "Regular" }
[7 pts] Query:
USE cs122a;
SELECT DISTINCT a.title, a.quality, a.popularity
FROM Article a, Has_read h
WHERE a.articleid = h.article_articleid AND a.userid = h.article_userid
AND (h.like_score = 3 OR h.like_score = 4);

[3 pts] Result:
{ "title": "Inauguration of President Trump", "quality": "Clean", "popularity": "Regular" }
{ "title": "Cleveland Cavs signed Kevin Durant", "quality": "Suspicious", "popularity": "Regular" }
{ "title": "2009 Warmest Year on Record", "quality": "Junk", "popularity": "Regular" }
{ "title": "iPhone 8 to be announced in September", "quality": "Clean", "popularity": "Regular" }

10. [10 pts] Write a query that lists the top 3 detectors that have reported articles (i.e., the top 3 detectors based on their report counts) by printing their userid along with their report counts in descending order. [Result size: 3]
Sample output: { "userid": 22, "cnt": 3 }
[7 pts] Query:
USE cs122a;
SELECT r.userid, COUNT(*) AS cnt
FROM Reports r
GROUP BY r.userid
ORDER BY cnt DESC
LIMIT 3;

[3 pts] Result:
{ "userid": 22, "cnt": 3 }
{ "userid": 15, "cnt": 3 }
{ "userid": 17, "cnt": 3 }

[EXTRA CREDIT]

11. [20 pts] Come up with an interesting/creative question of your own about this collection of data --
something that was not asked in the previous SQL-based assignments -- 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: