

Last Name: SUH

First Name: JOOWON

Student ID: 44414081

1. [10 pts] Take a look at the load script you ran in the instructions (`checkedtweets_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'd 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.*)

1NF: EvidenceFrom

Non-1NF: RawTweet (Coordinate, Hashtag), User ( expertise, phone), Evidence ( about), Verification (Evidence)

To be in 1NF each table the column should contain a single value, but the column in the parenthesis are declared as List, which means those can have multiple values.

2. [10pts] Repeat **step 0** (the setup/loading step) from the instructions document, but this time use the **schemaless** version of CheckedTweets.org. The schemaless version is the load script named `checkedtweets_schemaless.sqlpp`. The data file is the same one as in step 0, but you will need to change the 'USE' statement at the top of the file to say **USE checkedtweets\_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 `checkedtweets_schema` while the schemaless one is named `checkedtweets_schemaless`.)

The DDL for schemaless version defines only one type while schema version is defining all the types of attributes. Additionally, DDL for schemaless is automatically generating the tables, so the types of attributes will be defined depends on the data types in the data. The result of the query is still exactly same.

```
{ "tweetCount": 20000, "userCount": 3000, "evidenceFromCount": 1706, "evidenceCount": 260,
"verificationCount": 703 }

{ "tweetCount": 20000, "userCount": 3000, "evidenceFromCount": 1706, "evidenceCount": 260,
"verificationCount": 703 }
```

3. [10 pts] Looking back at the E-R diagram from the CheckedTweets.org conceptual design, compare and contrast the relational (MySQL) 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. To help make it clear how things have been mapped for SQL++/AsterixDB, the schema DDL version includes comments that indicate which E-R attributes correspond to which data in the raw JSON tweets. (And of course you may also recall that information from your trigger-writing work a few HW's ago.) 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 User u LIMIT 20;"` and similarly for Verification), how we have captured the information from each of the following E-R entities differently in AsterixDB and what the benefit(s) of this new design probably are:

**User:** *The phone, checker, and expertise are inside of User. In MySQL those are defined individual table. This means, in AsterixDB IsA relation has absorbed into the User table.*

**Verification:** *There is no VerifiedUsing relation in AsterixDB DDL. Instead Verification in AsterixDB has verified\_by and evidence.*

*This not only means the relation has merged with table, but also means that multivalued attributes has merged into table. In my opinion, this can benefit in efficiency. Let's say we want to query to look up the checkers in the system. And there are 1000 users with half of them are Checker. With old design system we have to check the equality of User.id = Checker.id. For this, old system should compare 1000\*500 times to find the checkers. On the other hand, with new design we only have to check if the kind attribute in User is Checker or User. So, new design system will compare only 1000 times. Similarly, to find the evidence that supports tweet. Old system need to compute # of Verification \* # of verifiedUsing while new design only needs to compute # of Verification.*

4. [7 pts] Use the *dataverse checkedtweets\_schema* for problems 4 and beyond. For checkers with the first name “Christopher” and the last name “Smith”, list their user ids, their addresses, and the dates they started being a checker. (Hint: A checker is a user of the *kind* “CHECKER”). [Result size: 2]

Sample output:

```
{ "user_id": 2401, "address": { "country": "United States of America",  
  "state": "WI", "city": "Larsen" }, "checker_since": "2020-01-  
20T20:43:20.000Z" }
```

[4 pts] Query:

```
select u.user_id, u.address, u.checker_since  
from User u  
where u.kind = "CHECKER" and u.name.first = "Christopher" and u.name.last = "Smith";
```

[3 pts] Result:

```
{ "user_id": 2401, "address": { "country": "United States of America", "state": "WI", "city": "Larsen" },  
  "checker_since": "2020-01-20T20:43:20.000Z" }  
  
{ "user_id": 181, "address": { "country": "United States of America", "state": "AR", "city": "Horseshoe Bend" },  
  "checker_since": "2020-08-14T21:26:42.000Z" }
```

5. [13 pts] For users that live in the city “Upham” or “East Megan”, write a query to get the users’ user\_id, full name, email, and the evidence URLs they have submitted. [Result size: 2]

Sample output:

```
{ "user_id": 136, "first": "Nathaniel", "last": "Schneider", "email":  
"schneider.nat@gmail.com", "urls": [ "https://www.healthy-habits.fr" ] }
```

[7 pts] Query:

```
select u.user_id, u.name.first, u.name.last, u.email, e.url  
from User u, EvidenceFrom ef, Evidence e  
where u.user_id = ef.user_id and ef.ev_id = e.ev_id and ( u.address.city = "Upham" or u.address.city =  
"East Megan" );
```

[3 pts] Result:

```
{ "user_id": 21, "first": "Cynthia", "last": "Stein", "email": "Stein.cynthia47@gmail.com", "url": "https://health.kh" }  
  
{ "user_id": 136, "first": "Nathaniel", "last": "Schneider", "email": "schneider.nat@gmail.com", "url":  
"https://www.healthy-habits.fr" }
```

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

Yes, it worked. The result was same. It tells me that both are making same dataset.

6. [10 pts] Write a query to print the tweet\_ids and their corresponding hashtag texts for tweets where the number of hashtags used in the tweet is greater than 23. [Result size: 1]

**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 :-))

A similar output:

```
{ "id": "1321194418041458688", "ht_list": [ "voting", "vote", ...,
"COVID19" ]}
** That's not the output, but it's what your output should look like :-)
```

[7 pts] Query:

```
select value {
  "id":t.id,
  "ht_list": (select value ht.text
from RawTweet t, t.extended_tweet.entities.hashtags ht
let c = array_count(t.extended_tweet.entities.hashtags)
where c > 23)
}
from RawTweet t
where array_count( t.extended_tweet.entities.hashtags) > 23;
```

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

It is not, since the ht\_list has more than 1 value, which violates the rule of the 1NF.

[1 pts] Result:

```
{ "id": "1321199733621379076", "ht_list": [ "Trump", "GOP", "Georgia", "Florida", "Texas", "Alaska", "Arizona",
"Pennsylvania", "Michigan", "Ohio", "Maine", "Louisiana", "Mississippi", "SouthCarolina", "NorthCarolina",
"Kansas", "Tennessee", "Kentucky", "Wisconsin", "Minnesota", "Colorado", "Missouri", "Oklahoma", "Idaho",
"VoteBlue" ] }
```

7. [10 pts] Write a query that prints the tweet\_ids and a list of ver\_ids for those tweets that have been posted after "2020-08-10 00:00:00", that have been verified more than once (i.e., the size of ver\_ids is greater than 1), **and** that also contain the hashtag COVID19 (all uppercase letters). [Result size: 4]

**Hints:**

- Check out the lecture and docs about the existential [SOME](#) clause. You might find it useful. :-)

Sample output:

```
{ "id": "1321210168722468864", "ver_ids": [ 660, 80 ] }
```

[7 pts] Query:

```
select t3.id, t3.ver_ids
from(
  Select value{
    "id":t2.id,
    "ver_ids": (select value v.ver_id
                from Verification v
                where v.tweet_id = t2.id),
    "ht_list":(select value ht.text
              from RawTweet t, t.extended_tweet.entities.hashtags ht)
  }
  From (select t1.id
        from(
          select t.id, count(*) as cnt
          from Verification v, RawTweet t
          where v.tweet_id = t.id and t.created_at > "2020-08-10 00:00:00"
          group by t.id ) as t1
        where t1.cnt>=2) as t2
  )as t3
where some h in t3.ht_list satisfies h = "COVID19";
```

[3 pts] Result:

```
{ "id": "1321199770656976903", "ver_ids": [ 280, 26 ] }
{ "id": "1321202970898251776", "ver_ids": [ 46, 401 ] }
{ "id": "1321205651348082693", "ver_ids": [ 61, 490 ] }
{ "id": "1321207379313250304", "ver_ids": [ 66, 546 ] }
{ "id": "1321210168722468864", "ver_ids": [ 80, 660 ] }
```

8. [10 pts] Write a query that analyzes hashtag popularity. It should print all of the **normalized** hashtags (e.g., Covid19, CoViD19 and coviD19 should be normalized to covid19, and similarly for other hashtags) and the number of distinct Tweeters who used each hashtag. Order your result in descending order and limit the number of results to the top five. [Result size: 5, of course]

**Hints:**

- Check the UNNEST clause [here](#) and/or the shorthand for UNNEST as a join clause [here](#).

Sample output:

```
{ "hashtag": "vote", "cnt": 225 }
```

[7 pts] Query:

```
select lower(t1.text) as hashtag, count(*) cnt
from(
select ht.text
from RawTweet t, t.extended_tweet.entities.hashtags ht
)as t1
group by lower(t1.text)
order by count(*) desc
limit 5
```

[2 pts] Result:

```
{ "hashtag": "trump", "cnt": 298 }
{ "hashtag": "vote", "cnt": 259 }
{ "hashtag": "bidenharris2020", "cnt": 198 }
{ "hashtag": "biden", "cnt": 155 }
{ "hashtag": "election2020", "cnt": 149 }
```

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

Yes, because by normalizing the values each values became unique in the list of records. That is each record of the result is unique.

9. [10 pts] Write a query that returns the handles of tweeters and their number of Covid-tagged tweets for those tweeters who've used the hashtag “**covid19**” more than 3 times. Your query should normalize the hashtags to lowercase (e.g., **Covid19** should be converted to **covid19**) in order to properly consider all Covid-tagged tweets. [Result size: 2]

Sample output:

```
{ "cnt": 5, "handle": "ppl4justice" }
```

[7 pts] Query:

```
select t1.cnt, t1.handle
from(
select count(*) as cnt, t.user.screen_name as handle
from RawTweet t, t.extended_tweet.entities.hashtags ht
where lower(ht.text) = "covid19"
group by t.user.screen_name
)as t1
where t1.cnt >3
;
```

[1 pts] Result:

```
{ "cnt": 5, "handle": "ppl4justice" }
{ "cnt": 5, "handle": "CupofJoeintheD2" }
```


[2 pts] If you wanted to answer this question using the HW6 tables in MySQL, what are the tables that you would need? Do you find the SQL++ query easier or harder, and why? (Answer in ≤ 3 sentences.) Note that the last part of the question has no right answer. (We just want to get your opinion :-))

I need Tweeter, tweet, hash\_tags, and tweet\_id. SQL++ query easier since I could solve this problem with only one table. So, I could solve the problem with shorter code.

10. [10 pts] Write a query that, for those tweets that have been verified using more than 12 URLs, prints the tweets' texts and the URLs used to verify those tweets. Your result could have the same URL appear multiple times, i.e., not distinct (e.g., "<http://www.fact-checked.hr>" could appear twice in *urls*) and that is expected. [Result size: 2]



Sample output:

```
{ "urls": [ "https://minnesota-ballots.gov", ... "https://www.rhode-island-  
ballots.gov" ], "text": "@Nigel_Farage China Will Attack Russia In The 2020s  
Article  https://t.co/1xWWXIZDoR\n\n#Nxivm #COVID19...  
https://t.co/r6aeldNd6d" }
```

[7 pts] Query:

```
select value{  
  "urls":( select value e3.url  
    from(  
      select v.evidence  
      from(  
        select t1.cnt, t1.a  
        from(  
          select count(*) as cnt, a  
          from Evidence e, e.about a  
          group by a  
        )as t1  
        where cnt>12  
      ) as t2, Verification v, RawTweet t  
      where v.tweet_id = t2.a and t.id = t2.a  
    ) as t3, t3.evidence e2, Evidence e3  
    where e2 = e3.ev_id),  
  "text":t3.text  
}  
from(  
  select v.evidence, t.text  
  from(  
    select t1.cnt, t1.a  
    from(  
      select count(*) as cnt, a  
      from Evidence e, e.about a  
      group by a  
    )as t1  
    where cnt>12  
  ) as t2, Verification v, RawTweet t  
  where v.tweet_id = t2.a and t.id = t2.a
```

as t3

;

[3 pts] Result:

```
{ "urls": [ "http://north-dakota-ballots.gov", "https://new-jersey-ballots.org", "http://www.tennessee-vote.gov",  
"https://www.arizona-election.org", "http://www.new-mexico-election.gov", "https://minnesota-ballots.gov",  
"http://www.ohio-election.gov", "https://utah-vote.org", "http://new-hampshire-ballots.org", "https://alabama-  
election.gov", "http://www.north-carolina-vote.org", "https://www.colorado-vote.org",  
"http://www.pennsylvania-ballots.gov", "http://illinois-election.gov", "https://www.south-dakota-election.gov",  
"https://www.georgia-election.org", "https://vermont-vote.gov", "http://alaska-ballots.gov",  
"https://www.colorado-vote.org", "https://www.maryland-election.org", "http://www.pennsylvania-ballots.gov",  
"https://www.montana-election.gov", "https://alabama-election.gov", "https://louisiana-ballots.gov" ], "text":  
"Why is #trump spreading false lies about #JoeBiden How come trump is not talking about what he will do for the  
coun... https://t.co/yD981X7YPu" }
```

```
{ "urls": [ "http://north-dakota-ballots.gov", "https://new-jersey-ballots.org", "http://www.tennessee-vote.gov",  
"https://www.arizona-election.org", "http://www.new-mexico-election.gov", "https://minnesota-ballots.gov",  
"http://www.ohio-election.gov", "https://utah-vote.org", "http://new-hampshire-ballots.org", "https://alabama-  
election.gov", "http://www.north-carolina-vote.org", "https://www.colorado-vote.org",  
"http://www.pennsylvania-ballots.gov", "http://illinois-election.gov", "https://www.south-dakota-election.gov",  
"https://www.georgia-election.org", "https://vermont-vote.gov", "http://alaska-ballots.gov",  
"https://www.colorado-vote.org", "https://www.maryland-election.org", "http://www.pennsylvania-ballots.gov",  
"https://www.montana-election.gov", "https://alabama-election.gov", "https://louisiana-ballots.gov" ], "text":  
"#Biden Says #Trump pick Barrett Supreme Court Confirmation Threatens #ACA - Newsweek 🗳️ #election2020  
#Texas... https://t.co/wcxIwD6POM" }
```