Homework 2

Suggestions on how to reproduce my results.

1. Please first inspect q[1|3]_loaddata.sql and q2_make.sql and replace the csv files with the correct ones on your machine.
2. Please refer to run.sh for the detailed commands I used to product the results.
3. See https://github.com/sunyinqi0508/dbhw2 for query outputs and updated documentations.
4. I used sql profiling to get the execution time for each query. However, profiling can negatively impact execution time. So, please turn it off and use the mysql prompt instead

Environment: MacBookPro11,3 (Late 2013, i7-4960hq) on macOS. MySQL 8.0.27.

Question 1

See typescript for the terminal outputs. And out1.txt for the query results. The execution times acquired by 'show profiles;' are appended at the end of out1.txt. Results:

Query_ID	Duration
1	17.89631700
2	29.54815600
3	40.23550600
4	29.61471900

Question 2

2.1 Reduce Unnecessary DISTINCT

In this experiment, I used the table schema from question 1 and tested on 2 simple selection queries. The first will select unique pairs of stocksymbol, time and the second will simply select every pairs of stocksymbol, time. Because 'time' is unique for each record, the result of two queries are identical.

On MySQL, I first tested on the data from Question 1. The query with distinct is significantly slower than the one without. Then I used the changed the datatype of stocksymbol from varchar(15) to decimal which is much faster in hashing. The gap between the two queries narrowed significantly. Finally, I theorized that denser data may be faster in hashing, therefore I used 'datagen .3 50 10000000' so that stocksymbol is much denser. The execution time for distinct query decreased by 40%.

On Aquery, I observed the same performance difference between queries with distinct and the one without. However, the improvement on changing datatype is less noticeable. I think this may be because AQuery/kdb uses different approach on DISTINCT. Also density of stocksymbol don't seem to be improve the performance of distinct.

2.2.1 Indexing (MySQL only)

A covering index will significantly accelerate the query execution than a non-covering index. In the first pairs of the predicates from where clause are not covered by the index and it's updating the indexed columns causing frequent index update. Therefore, indexed table will perform much worse. (10x)
However, the second pairs of query is covered by the index and doesn't updating any indexed columns, therefore the indexed performance is 10x better than the unindexed.

2.2.2 Denormalization (AQuery only)

Because the 'xkey' in AQuery/kdb doesn't seem to be used in accelrating query execution, and I didn't find a easy way to emulate index behavior, I'll just use the denormalization experiment instead.
In the first query, table are denormalized (joined in advance), the second query was joined on the fly. When the join condition is on a pk-fk. therefore denormalized table is not much bigger. It performs better (first experiments). denormalized table is much bigger, denormalization will perform worse.

Question 3

This query may be largely impacted by profiling. So, please turn off profiling for the last query and use the time from mysql prompt instead.

```
| Query_ID | Duration    | Query
|       38 |  0.00704100 | create table unique_likes(person decimal(9), artist decimal(9), UNIQUE KEY (person, artist))
|       39 |  0.99799100 | insert into unique_likes(person, artist) select distinct person, artist from likes order by
|       40 |  0.00518500 | create table unique_friend(person1 decimal(9), person2 decimal(9))
|       41 | 13.64329300 | insert into unique_friend(person1, person2)
select distinct person1, person2  from friend
where person1 < person2 or (person2, person1) not in (select person1, person2 from friend) order by person1, person2
|       42 |  1.14198800 | create unique index idx_f on unique_friend(person2, person1)
|       43 |  0.14631700 | create index idx_p1 on unique_likes(person)
```

For the last query I turned off profiling and used mysql prompt, otherwise it will take about 40 seconds.:
7308105 rows in set (22.07 sec)
Total execution time: 38.01181500