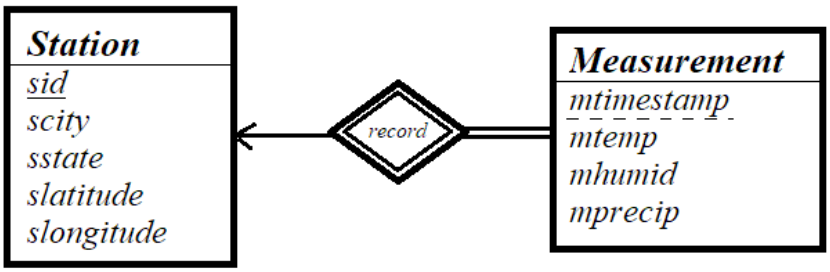
NYU Tandon School of Engineering
Computer Science and Engineering
CS6083, Fall 2019

Oct. 28, 2019

Problem Set #2 Sample Solution

Problem 1:
(a)



Weak entity: Measurement

(b) omitted

(c)
(i)

```
SELECT scity, AVG(mtemp)
FROM station NATURAL JOIN measurement
WHERE mtimestamp >= '2018-03-01 00:00:00' AND mtimestamp <= '2018-03-31 23:59:59'
GROUP BY scity;
```

scity	AVG(mtemp)
Berkeley	39.6770
Buffalo	76.1140
Glendale	39.3985
Lincoln	62.5714
Miami	44.0698
New York	37.2930
Orlando	43.6117
Philadelphia	49.1862
Pittsburgh	45.8569
Providence	59.9442
Rochester	50.4229
Warwick	56.9091

(ii)

```
CREATE VIEW max_feb_tem AS
SELECT sstate, MAX(mtemp) AS max_mtemp
FROM station NATURAL JOIN measurement
WHERE MONTH(mtimestamp) = 2
GROUP BY sstate;

SELECT DISTINCT max_feb_tem.sstate AS state, max_mtemp AS temp, scity AS city
```

```
FROM station NATURAL JOIN measurement, max_feb_tem
WHERE station.sstate = max_feb_tem.sstate AND measurement.mtemp = max_feb_tem.max_mtemp;
```

state	temp	city
Pennsylvania	49	Philadelphia
Pennsylvania	49	Pittsburgh
Florida	41	Miami
Florida	41	Orlando
New York	82	Buffalo
California	36	Berkeley
California	36	Glendale
Rhode Island	71	Providence
Rhode Island	71	Lincoln

(iii)

```
CREATE VIEW adjacent_time_pairs AS
SELECT prevm.sid AS sid, prevm.mtimestamp AS prevts, MIN(nextm.mtimestamp) AS nextts
FROM measurement AS prevm, measurement AS nextm
WHERE prevm.sid = nextm.sid AND prevm.mtimestamp < nextm.mtimestamp
GROUP BY prevm.sid, prevm.mtimestamp;

SELECT
        pair.sid AS sid,
        SUM(prevm.mhumid < nextm.mhumid) / COUNT(*) AS humidity_also_rise_likelihood
FROM adjacent_time_pairs AS pair, measurement AS prevm, measurement AS nextm
WHERE
        pair.sid = prevm.sid AND pair.sid = nextm.sid
        AND pair.prevts = prevm.mtimestamp AND pair.nextts = nextm.mtimestamp
        AND prevm.mtemp < nextm.mtemp
GROUP BY pair.sid;
```

sid	humidity_also_rise_likelihood
5	0.8393
6	0.8171
7	0.8548
8	0.8147
9	0.8935
10	0.8889
11	0.8390
12	0.8018
13	0.8451
14	0.8428
15	0.8991
16	0.8248
17	0.8136
18	0.8243

(iv)

```
SELECT s1.sid, s2.sid,
110.25 * SQRT(
        (s1.slatitude - s2.slatitude) * (s1.slatitude - s2.slatitude) +
        (s1.slongitude - s2.slongitude) * (s1.slongitude - s2.slongitude) * COS(s2.slatitude) *
COS(s2.slatitude)
```

```
) AS distance
FROM station AS s1, station AS s2
WHERE s1.sid < s2.sid AND s1.sstate = 'Rhode Island' AND s2.sstate = 'Rhode Island';
```

sid	sid	distance
15	16	9.959207846694108
15	17	14.331350379066793
15	18	23.491780827734136
16	17	4.413026962980212
16	18	13.575071413483803
17	18	9.400619144105333

(v)

```
DELIMITER $$
CREATE FUNCTION `weather_station`.`distance`(lat1 FLOAT, lng1 FLOAT, lat2 FLOAT, lng2 FLOAT)
   RETURNS FLOAT
   BEGIN
        DECLARE dist FLOAT;
        SET dist = 110.25 * SQRT(
                (lat1 - lat2) * (lat1 - lat2) +
                (lng1 - lng2) * (lng1 - lng2) * COS(lat2) * COS(lat2)
        );
        RETURN dist;
   END$$
DELIMITER ;
```

(vi)

```
CREATE VIEW total_annual_rainfall AS
SELECT sid, YEAR(mtimestamp) AS yyyy, SUM(mpriecip) AS total
FROM station NATURAL JOIN measurement
GROUP BY sid, YEAR(mtimestamp);

CREATE VIEW average_total_annual_rainfall AS
SELECT sid, AVG(total) AS rainfall
FROM total_annual_rainfall
GROUP BY sid;

SELECT s1.sid, s1.scity, s2.sid, s2.scity
FROM station AS s1, station AS s2, average_total_annual_rainfall AS r1, average_total_annual_rainfall
AS r2
WHERE
        s1.sid < s2.sid
        AND s1.sid = r1.sid AND s2.sid = r2.sid
        AND ABS(r1.rainfall - r2.rainfall) >= 50
        AND weather_station.distance(s1.slatitude, s1.slongitude, s2.slatitude, s2.slongitude) <= 16.09;
```

sid	scity	sid	scity

(empty)

Note that the output here is empty as the value 50 is too large. For smaller values, say 1 or 5, you would get more output. Also, the value 16.09 in the above query is 10 miles in kilometers.

(d)

(i)

$$_{scity}\mathcal{G}_{avg(mtemp)}\left(\sigma_{year(mtimestamp)=2018 \wedge month(mtimestamp)=3}(Station \bowtie Measurement)\right)$$

(ii)

$$maxtemp \leftarrow {}_{sstate}\mathcal{G}_{max(mtemp)\ as\ maxtemp}\left(\sigma_{month(mtimestamp)=2}(Station \bowtie Measurement)\right)$$

$$\Pi_{sstate,\ maxtemp,city}\left(\sigma_{Station.sstate=maxtemp.sstate \wedge Measurement.mtemp=maxtemp.maxtemp}(Station \bowtie Measurement \times maxtemp)\right)$$

(iii)

$$pairs \leftarrow \sigma_{m1.sid=m2.sid \wedge m1.mtimestamp=m2.mtimestamp}\left(\rho_{m1}Measurement \times \rho_{m2}Measurement\right)$$

$$tspairs \leftarrow {}_{m1.sid\ as\ sid,m1.mtimestamp\ as\ prevts}\mathcal{G}_{min(m2.mtimestamp)\ as\ nextts}(pairs)$$

$$temppairs \leftarrow$$

$$\sigma_{tspairs.sid=prevm.sid \wedge tspairs.sid=nextm.sid \wedge tspair.prevts=prevm.mtimestamp \wedge tspair.nextts=nextm.mtimestamp}\left(\rho_{prevm}Measurement \bowtie \rho_{nextm}Measurement \times tspairs\right)$$

$$_{sid}\mathcal{G}_{sum(prevm.mhumid < nextm.mhumid) \div count(*)\ as\ likelihood}(temppairs)$$

(iv)

$$pairs \leftarrow \sigma_{s1.sid<s2.sid \wedge s1.sstate='Rhode\ Island' \wedge s2.sstate='Rhode\ Island'}\left(\rho_{s1}Station \times \rho_{s2}Station\right)$$

$$\Pi_{s1.sid,s2.sid,110.25 \times \sqrt{(s1.slatitude - s2.slatitude)^2 + \left((s1.slongitude - s2.slongitude) \times \cos(s2.latitude)\right)^2}}(pairs)$$

(e)

(i)

```
INSERT INTO station(scity, sstate, slatitude, slongitude) VALUES
('Austin','Texas', 30.3079827, -97.893485);
```

(ii)

```
DELETE FROM measurement WHERE sid = 1234;
DELETE FROM station WHERE sid = 1234;
```

(iii)

```
UPDATE measurement SET mtemp = NULL WHERE mtemp > 120 OR mtemp < -60;
```

Problem 2

(a)

```
CREATE VIEW city_data AS (
        SELECT
                AVG(mtemp) AS temp,
                AVG(mhumid) AS humid,
                SUM(mpriecip) AS precip,
```

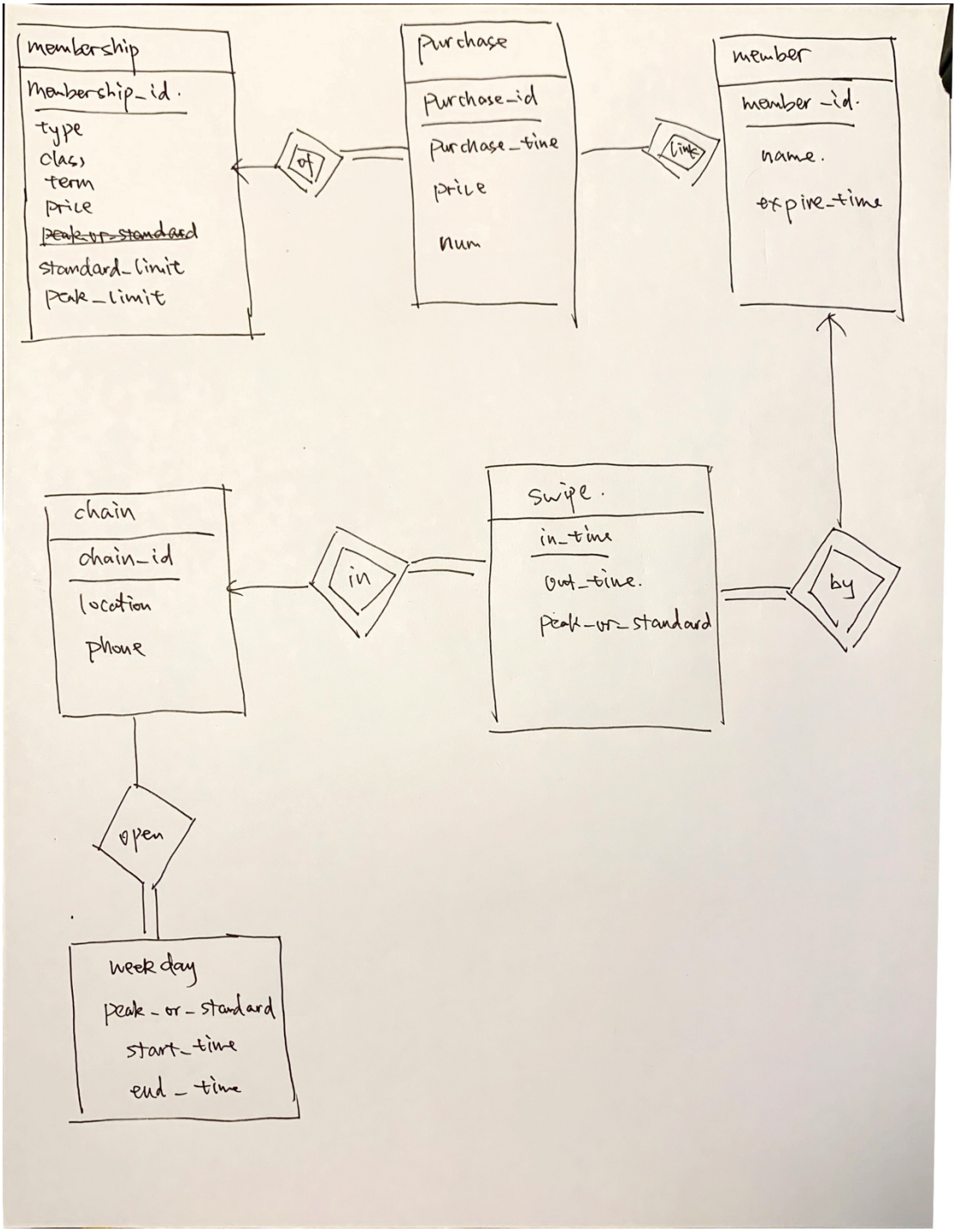
```
                scity AS city,
                DATE(mtimestamp) AS mdate
        FROM station NATURAL JOIN measurement
        GROUP BY scity, DATE(mtimestamp)
);

(b)
SELECT cd1.city, cd1.mdate
FROM city_data AS cd1, (
        SELECT MAX(temp) AS max_temp, mdate
        FROM city_data
        WHERE YEAR(mdate) = 2018
        GROUP BY mdate
) AS cd2
WHERE cd1.temp = cd2.max_temp AND cd1.mdate = cd2.mdate;

(c)
DELIMITER $$
CREATE TRIGGER `weather_station`.`reject_extreme` BEFORE INSERT
  ON `weather_station`.`measurement`
  FOR EACH ROW BEGIN
        DECLARE prevtemp INT;
        DECLARE prevavg  INT;

        SELECT mtemp INTO prevtemp
        FROM measurement AS m1,
        (
                SELECT MAX(mtimestamp) AS max_ts
                FROM measurement
                WHERE sid = new.sid
        ) AS m2
        WHERE m1.sid = new.sid AND m1.mtimestamp = m2.max_ts;

        SELECT AVG(mtemp) INTO prevavg
        FROM measurement
        WHERE sid = new.sid
                AND MONTH(new.mtimestamp) = MONTH(mtimestamp)
                AND DAY(new.mtimestamp) = DAY(mtimestamp)
                AND YEAR(new.mtimestamp) - YEAR(mtimestamp) <= 10;

        IF (
                (prevtemp IS NOT NULL AND ABS(new.mtemp - prevtemp) > 10)
                AND (prevavg IS NOT NULL AND ABS(new.mtemp - prevavg) > 10)
        ) THEN
                SIGNAL SQLSTATE '45000';
        END IF;
  END$$
```

```
DELIMITER ;

(d)
/* create table */
CREATE TABLE `tempRecords` (
	`city` char(100) NOT NULL,
	`month` int(11) NOT NULL,
	`day` int(11) NOT NULL,
	`temperature` int(11) DEFAULT NULL,
	`occur_year` int(11) DEFAULT NULL
)

/* initialize new table data from existing data*/
INSERT INTO tempRecords (city, MONTH, DAY, temperature, occur_year)
SELECT DISTINCT ms2.scity, ms2.mm, ms2.dd, ms2.max_temp, YEAR(m1.mtimestamp)
FROM measurement m1 NATURAL JOIN station s1,
(
	SELECT scity, MONTH(mtimestamp) AS mm, DAY(mtimestamp) AS dd, MAX(mtemp) AS
max_temp
	FROM measurement NATURAL JOIN station
	GROUP BY scity, MONTH(mtimestamp), DAY(mtimestamp)
) AS ms2
WHERE MONTH(m1.mtimestamp) = ms2.mm AND DAY(m1.mtimestamp) = ms2.mm AND s1.scity =
ms2.scity AND m1.mtemp = ms2.max_temp;

/* trigger */
DELIMITER $$
CREATE
   TRIGGER `weather_station`.`track_max_temp` AFTER INSERT
   ON `weather_station`.`measurement`
   FOR EACH ROW BEGIN
	DECLARE cur_city CHAR(100);
	DECLARE cur_temp INT;
	DECLARE cur_max_count INT;

	SELECT scity INTO cur_city
	FROM station
	WHERE sid = new.sid;

	SELECT MAX(temperature) INTO cur_temp
	FROM tempRecords
	WHERE city = cur_city
		AND MONTH = MONTH(new.mtimestamp)
		AND DAY = DAY(new.mtimestamp);

	IF (cur_temp = new.mtemp) THEN
```

```
            /* check if already exists */
            SELECT COUNT(*) INTO cur_max_count
            FROM tempRecords
            WHERE city = cur_city
                AND MONTH = MONTH(new.mtimestamp)
                AND DAY = DAY(new.mtimestamp)
                AND temperature = cur_temp
                AND YEAR(new.mtimestamp) = occur_year;
            IF cur_max_count = 0 THEN
                INSERT INTO tempRecords (city, MONTH, DAY, temperature, occur_year)
                VALUES (cur_city,
                    MONTH(new.mtimestamp),
                    DAY(new.mtimestamp),
                    new.mtemp,
                    YEAR(new.mtimestamp)
                );
            END IF;
        ELSEIF (cur_temp < new.mtemp) THEN
            /* delete prev and insert new*/
            DELETE FROM tempRecords
            WHERE city = cur_city
                AND MONTH = MONTH(new.mtimestamp)
                AND DAY = DAY(new.mtimestamp)
                AND temperature = cur_temp;

            INSERT INTO tempRecords (city, MONTH, DAY, temperature, occur_year)
            VALUES (cur_city,
                MONTH(new.mtimestamp),
                DAY(new.mtimestamp),
                new.mtemp,
                YEAR(new.mtimestamp)
            );
        END IF;
    END$$
DELIMITER ;
```

3：

A.


membership
membership_id.
type
class
term
price
~~peak_or_standard~~
standard_limit
peak_limit
of
Purchase
Purchase_id
Purchase_time
price
num
link
member
member_id.
name.
expire_time
chain
chain_id
location
phone
in
swipe.
in_time
out_time.
peak_or_standard
by
open
weekday
peak_or_standard
start_time
end_time

B.

Membership(**membership_id**, type, class, term, price, peak_limit, standard_limit)

Chain(**chain_id**, location, phone)

Purchase(**purchase_id**, *merbership_id, purchase*_time, price, num)

Member(**member_id**, name, expire_time)

Link(***member_id, purchase_id***)

Swipe(**member_id, in_time**, out_time, **chain_id**, peak_or_standard)

Schedule(**chain_id, weekday, peak_or_standard**, start_time, end_time,)

member_id in Link references Member.member_id

purchase_id in Line references Purchase.purchase_id

member_id in Swipe references Member.member_id

chain_id in Swipe references Chain.chain_id

chain_id in Schedule references Chain.chain_id

assumption:

The expire_time in member table will be updated when this member links the new purchase

C

1.

Select avg(price). // or avg(price / num)

From Purchase natural join Membership

Where purchase_time + month(num * int(term))>= “20160401”

2.

Select chain_id, count(*)

From Chain natural join Swipe

Where date(in_time) = “20180923” and date(out_time) = “20190923” and time(in_time) <= “19:30” and time(out_time) >= “19:30”

Group by chain_id

3.

```
Select chain_id
From Schedule
Where peak_or_standard = “standard” and 16 >= hour(start_ime) and 17 <=hour(end_time)
Group by chain_id
Having count(*) = 7
```

// remove the last two lines if the question means that we need query the chain only if it open one day during 4 - 5pm

4.

Note: People have multiple membership, when people use the facilities, it will consume all the membership time limit

```
With MembershipValidInAugust as (
        Select purchase_id
        From Purchase natural join Membership
        Where date(purchase_time) + month(num * int(term)) >= “20190801”),
With StandardTimeUsage as (
        Select purchase_id, sum(out_time – in_time) as usage
        From MembershipValidInAugust natural join Swipe
        Where peak_or_standard = “standard”
Select purchase_id, usage, count(*) as num
From StandardTimeUsage natural join Link
Group by purchase_id
```