NYU Tandon School of Engineering **March 9, 2017**
Computer Science and Engineering
CS6083, Spring 2017

Problem Set #2

Problem 1:

a) Answer:
- No, it is not a good idea. Having only cusid, carid as primary key means one customer cannot take the same car in another trip which will not be practical and will create conflict in the database. For example, if customer A took car B yesterday, and he take the same car today, the key cannot identify these two tuples and store them in the schema the same time.
- Yes, it would work. It can identify a trip record, since a driver cannot drive two cars in one trip so we do not need to use carid as part of the primary key.

b) Answer:
- CarOwnership.did is a foreign key referencing Driver.did
- CarOwnership.carid is a foreign key referencing Car.carid
- Trips.cusid is a foreign key referencing Customer.cusid
- (Trip.did, Trip.carid) is a foreign key referencing (CarOwnership.did, CarOwnership.carid)

c) Answer:

```
CREATE TABLE `Customer` (
    `cusid` INT NOT NULL,
    `cusname` VARCHAR(45) NOT NULL,
    `cusphone` VARCHAR(45) NOT NULL,
    `cuscity` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`cusid`));

CREATE TABLE `Driver` (
     `did` INT NOT NULL,
    `dname` VARCHAR(45) NOT NULL,
    `dphone` VARCHAR(45) NOT NULL,
    `dcity` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`did`));

 CREATE TABLE `Car` (
    `carid` INT NOT NULL,
    `carbrand` VARCHAR(45) NOT NULL,
    `carsize` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`carid`));

CREATE TABLE `CarOwnership` (
```

```
    `did` INT NOT NULL,
    `carid` INT NOT NULL,
    PRIMARY KEY (`did`, `carid`),
    FOREIGN KEY (`did`) REFERENCES `Driver` (`did`),
    FOREIGN KEY (`carid`) REFERENCES `Car` (`carid`));

CREATE TABLE `Trips` (
    `cusid` INT NOT NULL,
    `carid` INT NOT NULL,
    `did` INT NOT NULL,
    `getontime` DATETIME NOT NULL,
    `getofftime` DATETIME NULL,
    `price` FLOAT NULL,
    `distance` FLOAT NULL,
    PRIMARY KEY (`cusid`, `carid`, `did`, `getontime`),
    FOREIGN KEY (`cusid`) REFERENCES `Customer` (`cusid`),
    FOREIGN KEY (`carid`, `did`) REFERENCES `CarOwnership` (`carid`, `did`));
```

d) Answer:

(i)
```
SELECT cusid, cusname
FROM CUSTOMER
WHERE cusid in(
        SELECT cusid
        FROM Trips
        WHERE YEAR(getontime) = '2017' and MONTH(getontime) = '1')
```

(ii)
```
SELECT cusid
FROM CUSTOMER
WHERE cuscity = 'Brooklyn' and cusid in(
        SELECT cusid
        FROM Trips
        WHERE did in(
                SELECT did
                FROM DRIVER
                WHERE dcity = 'Queens'))
```

(iii)
```
SELECT CUSTOMER.cusid, CUSTOMER.cusname
FROM CUSTOMER, Trips
WHERE CUSTOMER.cusid = Trips.cusid and
        Trips.price = (SELECT MAX(price) FROM Trips)
```

(iv)

```
With temp
As(
SELECT Car.carbrand, Count(distinct Trips.cusid) as cuscount
FROM Car, Trips
WHERE Car.carid = Trips.carid
GROUP BY Car.carbrand
)
SELECT temp.carbrand
FROM temp
WHERE temp.cuscount = (SELECT MAX(cuscount) FROM temp);
```

(v)

```
With temp
As(
SELECT did, SUM(price) as income
FROM Trips
WHERE YEAR(getontime) = '2017' and MONTH(getontime) = '1'
GROUP BY did
)
SELECT Driver.did, dname
FROM Driver, temp
WHERE Driver.did = temp.did and
        income = (SELECT MAX(income) FROM temp);
```

(vi)

```
With temp1
As(
SELECT cusid, SUM(distance) as total_dis
FROM Trips
GROUP BY cusid
)

With temp2
As(
SELECT Customer.cuscity, MAX(total_dis) as max_dis
FROM Customer, temp1
WHERE Customer.cusid = temp1.cusid
GROUP BY cuscity
)
SELECT Customer.cuscity, Customer.cusid
```

```
FROM Customer, temp1, temp2
WHERE Customer.cusid = temp1.cusid and Customer.cuscity = temp2.cuscity and
temp1.total_dis = max_dis;
```

(vii)

```
With temp1
as (
SELECT cuscity, month(getontime) as tmonth, count(*) as tripcount
FROM Trips, Customer
WHERE Trips.cusid = Customer.cusid and YEAR(getontime) = '2017'
GROUP BY cuscity, tmonth
)

With temp2
As(
SELECT tmonth, max(tripcount) as maxcount
FROM temp1
GROUP BY tmonth
)
SELECT temp1.tmonth, cuscity
FROM temp1, temp2
WHERE temp1.tmonth = temp2.tmonth and temp1.tripcount = temp2.maxcount;
```

(viii)
```
SELECT avg(timestampdiff(MINUTE, getontime, getofftime)) as avg_duration
FROM Driver, Trips
WHERE Driver.did = Trips.did and Driver.dcity = 'Brooklyn' and YEAR(Trips.getontime) =
'2017'
```

e) Answer:

(iii) temp ← $G_{\text{max(price) as maxprice}}$ (Trips)

$\prod_{\text{cusid, cusname}}(\sigma_{\text{Trips.price = temp.maxprice}}((\text{Trips} \times \text{temp}) \bowtie \text{CUSTOMER}))$

(iv) temp ← $_{\text{carbrand}}\,G_{\text{count(distinct Trips.cusid) as cuscount}}$ (Car ⋈ Trips)

temp2 ← $G_{\text{max(cuscount) as maxcount}}$ (temp)

$\prod_{\text{carbrand}}\sigma_{\text{temp.cuscount = temp2.maxcount}}$ (temp × temp2)

(v) temp ← $_{\text{did}}\,G_{\text{SUM(price) as income}}$ ($\sigma_{\text{year(getontime) = '2017' and month(getontime) = '1'}}$ (Trips))

temp2 ← $G_{\text{max(income) as maxincome}}$ (temp)

$\prod_{\text{Driver.did, dname}}(\sigma_{\text{temp.income = temp2.maxincome}}(\text{temp} \times \text{temp2}) \bowtie \text{Driver})$

(vi) temp1 ← $_{\text{cusid}}\,G_{\text{SUM(distance) as total\_dis}}$ (Trips)

temp2 ← $_{\text{cuscity}}\mathcal{G}_{\text{max(total\_dis) as max\_dis}}$ (Customer ⋈ temp1)
$\prod_{\text{Customer.cuscity, Customer.cusid}}$ $\sigma_{\text{temp1.total\_dis = temp2.max\_dis}}$ (temp ⋈ temp2 ⋈ Customer)

(vii) temp1 ← $_{\text{cuscity, month(getontime) as tmonth}}\mathcal{G}_{\text{count(*) as tripcount}}$ ($\sigma_{\text{year(getontime) = '2017'}}$ (Customer ⋈ temp1))
temp2 ← $_{\text{tmonth}}\mathcal{G}_{\text{max(tripcount) as maxcount}}$ (temp1)
$\prod_{\text{temp1.tmonth, cuscity}}$ $\sigma_{\text{temp1.tripcount = temp2.maxcount}}$ (temp1 ⋈ temp2)

(viii) $\mathcal{G}_{\text{avg(getofftime - getontime) as avg\_duration}}$ ($\sigma_{\text{dcity = 'Brooklyn' and year(getontime) = '2017'}}$ (Driver ⋈ Trips))

f) Answer:

(i)
```
UPDATE Car SET carsize = 'compact'
WHERE carsize = 'small'
```

(ii)
```
DELETE FROM Driver
WHERE did not in(
        SELECT did FROM CarOwnership
)
```

(iii)
```
UPDATE Trips SET price = price + 5
WHERE timestampdiff(MINUTE, getontime, getofftime) > 30
```

Problem 2:

(a) Answer:
```
CREATE VIEW TD
AS(
    SELECT Trips.cusid, cusname, cuscity, carid, did, getontime
    FROM Trips, Customer
    WHERE Trips.cusid = Customer.cusid
)
```

(i)

```
With temp1
As(
SELECT cusid, cusname, cuscity, count(*) as cuscount
FROM TD
GROUP BY cusid, cusname, cuscity
)

With temp2
As(
SELECT cuscity, max(cuscount) as maxcount
```

```
FROM temp1
GROUP BY cuscity
)
SELECT temp1.cuscity, cusid, cusname
FROM temp1, temp2
WHERE temp1.cuscity = temp2.cuscity and temp1.cuscount = temp2.maxcount;
```

(ii)
```
UPDATE TD SET cusname = 'Kety Perry'
WHERE cusname = 'Taylor Swift'
```

(b) Answer:

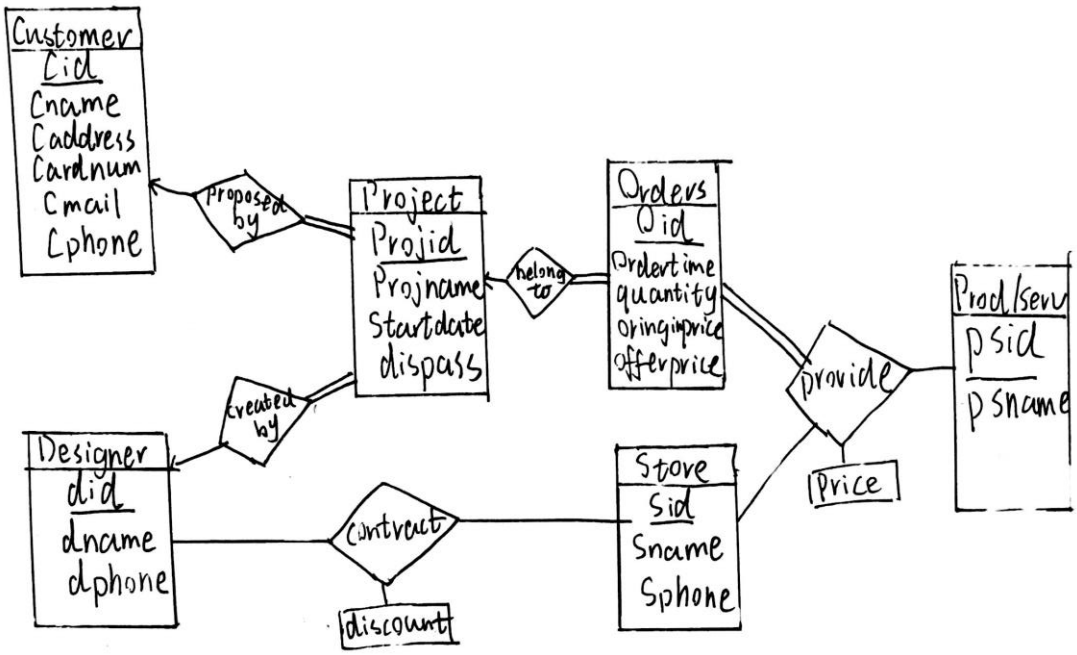
```
CREATE trigger autoCharge BEFORE INSERT
on Trips
FOR EACH ROW
BEGIN
    IF timestampdiff(MINUTE, NEW.getontime, NEW.getofftime) > 30 THEN
        SET NEW.price = NEW.price + 5;
    END IF;
END;
```

(c) Answer:

```
CREATE trigger DriverLimit BEFORE INSERT
on Trips
FOR EACH ROW
BEGIN
    IF (SELECT count(*)
        FROM Trips
        WHERE did = NEW.did and getofftime > (NEW.getontime - INTERVAL 60 MINUTE)
        ) >= 5 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = "Driver‘s trip limitation meets";
    END IF;
END;
```

Problem 3:

(a) Answer:



1. When the designer creates the project, he can decide if discount is passed to the customer. The decision will be related to all discount among the stores which have contracts with the designer.
2. A designer and a store can only hold at most one contract at the same time. If new discount is set, update the discount. If the contract is canceled, delete the record.
3. One project can have multiple purchase orders and one order would only contain one kind of product/service. In case the discount is updated, the afterprice would be a good choice to store old total price if discount is updated. Which price to pass is determined by the dispass in project.

(b) Answer:

Customer(<u>cid</u>, cname, caddress, cardnum, cemail, cphone)
Designer(<u>did</u>, dname, dphone)
Store(<u>sid</u>, sname, sphone)
ProdServ(<u>psid</u>, psname)
Provide(<u>sid</u>, <u>psid</u>, price)
Project(<u>projid</u>, cid, did, projname, startdate, dispass)
Contract(<u>did</u>, <u>sid</u>, discount)
Orders (<u>oid</u>, projid, psid, sid, ordertime, quantity, originprice, afterprice)

Customer(<u>cid</u>, cname, caddress, cardnum, cemail, cphone)
cid: the customer id
cname: the customer name
caddress: the customer address
cardnum: the credit card number

cemail: the customer email
cphone: the customer phone number
Primary key: cid

Designer(did, dname, dphone)
did: the designer id
dname: the designer name
dphone: the designer phone number
Primary key: did

Store(sid, sname, sphone)
sid: the store id
sname: the store name
sphone: the store phone number
Primary key: sid

ProdServ(psid, psname)
psid: the product/service id
psname: the product/service name
Primary key: psid

Provide(sid, psid, price)
sid: the store id
psid: the product id
price: the list price
Primary key: (sid, psid)
Foreign key: sid references Store.sid
psid references ProdServ.psid

Project(projid, cid, did, projname, startdate, dispass)
projid: the project id
cid: the customer id
did: the designer id
projname: the project name
startdate: the start date of the project
dispass: whether the discount is passed to customer. 0 means the designer keep it as commission. 1 means the designer pass it to the customer.
Primary key: projid
Foreign key: cid references Customer.cid
did references Designer.did
When the designer creates the project, he can decide if discount is passed to the customer. The decision will be related to all discount among the stores which have contracts with the designer.

Contract(did, sid, discount)
did: the designer id
sid: the store id
discount: the discount between designer and the store. It stores the percentage off.
Primary key: (did, sid)
Foreign key: did references Designer.did
sid references Store.sid
A designer and a store can only hold at most one contract at the same time. If new discount is set, update the discount. If the contract is canceled, delete the tuple.

Orders (oid, projid, psid, sid, ordertime, quantity, originprice, afterprice)
oid: the purchase order id
projid: the project id
psid: the product/service id
sid: the store id
ordertime: the ordered date
quantity: the quantity of the product purchased
originprice: the original price
afterprice: the price after discount (if there exist discount)
Primary key: oid
Foreign key: projid references Project.projid
(psid, sid) references (Provide.psid, Provide.sid)
One project can have multiple purchase orders and one order would only contain one kind of product/service. In case the discount is updated, the afterprice would be a good choice to store old total price if discount is updated. Which price to pass is determined by the dispass in project.

(c) Answer:

(i)
```
SELECT did, SUM(originprice - afterprice) AS totalcomm
FROM Project, Orders
WHERE Project.projid = Orders.projid and Project.dispass = 0
        and YEAR(Orders.ordertime) = '2016'
GROUP BY did
```

(ii)

```
With temp1
AS(
SELECT sid, discount, dispass
FROM Project, Contract
WHERE Project.did = Contract.did and projid = '82347'
)

With temp2
```

```
AS(
SELECT sid, price
FROM Provide
WHERE psid = '481787'
)

With temp3
AS(
SELECT temp2.sid, price, COALESCE(dispass, 0) as ifpass, ROUND(COALESCE(discount, 0),2)
as finaldis
FROM temp1 right join temp2 on (temp1.sid = temp2.sid);
)
SELECT temp3.sid, sname
FROM temp3, store
WHERE temp3.sid = store.sid and (price * (1-ifpass * finaldis)) =
        (SELECT MIN(price * (1-   ifpass * finaldis)) AS minprice FROM temp3);
```

If there is no discount contract or not pass, ifpass*finaldis = 0

(iii)

```
With temp1
AS(
SELECT Orders.projid, SUM(originprice*(1-dispass) + afterprice*dispass) as expense
FROM Orders, Project
WHERE Orders.projid = Project.projid
GROUP BY projid
)
SELECT Project.cid, cname, SUM(expense) as totalexpence
FROM temp1, Project, Customer
WHERE temp1.projid = Project.projid and Customer.cid = Project.cid
GROUP BY cid, cname;
```

The 1-dispass and dispass if to do a choice between originprice and afterprice. Dispass is either 0 or 1 as mentioned in assumption and defined type BIT(1).

(iv)

```
SELECT did
FROM Contract JOIN Store ON (Contract.sid = Store.sid)
WHERE discount = (SELECT max(discount)
                  FROM Contract, Store
                  WHERE Contract.sid = Store.sid and sname = 'Garden Paradise'
                 ) and sname = 'Garden Paradise'
```