1 MCQs

1. T/F: Given relations R and S , both with attribute A , $\pi_A(R \cap S) = \pi_A(R) \cap \pi_A(S)$
2. T/F: Given relations R and S , both with attribute A , $\pi_A(R \cup S) = \pi_A(R) \cup \pi_A(S)$
3. T/F: A trivial functional dependency is always of the form $A \to B$ where $B \subseteq A$.
4. T/F: Given $R = (A, B, C)$ and $A \to B$ and $B \to C$, the following decomposition is dependency preserving: $R_1 = (A, B), R_2 = (B, C)$.

1.1 Solution

1. False. Consider R = {(1, apple), (2, apple)}, S = {(1, banana), (2, banana)}, Projected onto the first elt first and then taking intersection returns {1, 2}. Taking intersection then projecting gets us nothing.
2. True. Unlike intersection, we don't lose any rows. So, this holds.
3. True. This is true by definition.
4. True. We don't need to do any joins to check for integrity.

2 Question 1

Consider the following schema:

```
sailor(id, name, rating, age)
boat(name, color)
reserves(s_id references sailor.id, b_name references boat.name, date)
```

Write the following queries in SQL:

1. Find the names of the sailors who have reserved at least one boat.
2. Find how many times the boat named "Buoyonce" has been reserved by each sailor. The result should not contain the sailors who have never reserved it.
3. Find the sailors who have reserved a red boat but not a green boat.
4. Find the sailors who have reserved all of the blue boats.

2.1 Solution

1. Find the names of the sailors who have reserved at least one boat.

```
SELECT s.name FROM Sailor AS s, Reserves AS r WHERE (s.id = r.s_id);
```

2. Find how many times the boat named "Buoyonce" has been reserved by each sailor. The result should not contain the sailors who have never reserved it.

```
SELECT s_id, COUNT(s_id) FROM Reserves WHERE (b_name = "Buoyonce") GROUP BY s_id;
```

3. Find the sailors who have reserved a red boat but not a green boat.

```
WITH RedBoats AS
    (SELECT r.s_id AS s_id FROM Boat AS b, Reserves AS r
        WHERE (b.color = "red" AND b.name = r.b_name)),
NotGreenBoats AS
    (SELECT r.s_id AS s_id FROM Boat AS b, Reserves AS r
        WHERE (b.color <> "green" AND b.name = r.b_name))
SELECT DISTINCT s_id FROM RedBoats AS rb, NotGreenBoats AS ngb
    WHERE (rb.s_id = ngb.s_id);
```

4. Find the sailors who have reserved all of the blue boats.

```
sql NOTE: No clue if the following is valid SQL WITH BlueBoats AS (SELECT r.s_id AS s_id FROM Boat AS b, Reserves AS r
 WHERE (b.color = "blue" AND b.name = r.b_name)), SELECT DISTINCT * FROM sailor WHERE (BlueBoats IN (SELECT b_name FROM
 sailor, reserves WHERE sailor.id = reserves.s_id));
```

3 Question 2

For the following E/R diagram, write an equivalent relational schema with the fewest number of relations. Indicate all keys and foreign keys.

3.1 Solution

```
person(id, name, address)
car(owner_id REFERENCES person.id, license, model, year)
accident(id, location, date)
participated(car_license REFERENCES car.license, accident_id REFERENCES accident.id, damage_amt)
```

4 Question 3

Given a schema R and a functional dependency set F as follows:

```
R = {A, B, C, D, E, Z}
F = {
    A -> BC,
    B -> CE,
    A -> E,
    AC -> Z,
    D -> B
}
```

1. Is D a superkey for R? Explain.
2. Is ABC a candidate key for R? Explain.
3. Is ABD a superkey for R? Explain using Armstrong's Axioms.
4. Show that the canonical cover of F, F_c is equal to the following:

```
Fc = {
    A -> BZ,
    B -> CE,
    D -> B
}
```

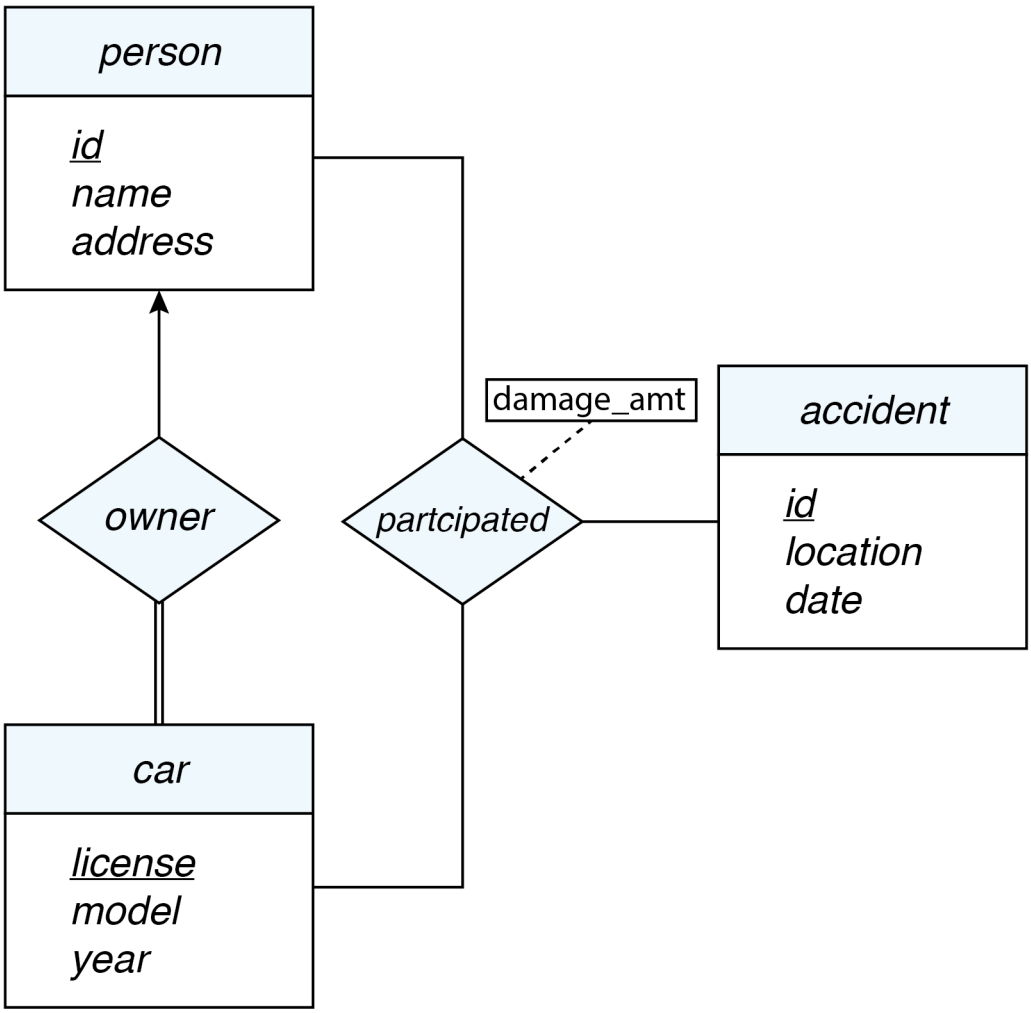
person
id
name
address
damage_amt
accident
owner
partcipated
id
location
date
car
license
model
year

Figure 1: Q2

4.1 Solution

1. Is D a superkey for R? Explain.

No. The set of attributes determined by D is {B, C, D, E}, which is not all of R. Thus, D is not a superkey.

2. Is ABC a candidate key for R? Explain.

No. Since A determines BC, A would also be a superkey, which means that ABC is not minimal, so it cannot be a candidate key.

3. Is ABD a superkey for R? Explain using Armstrong's Axioms.

We are given ABD D. Thus, by reflexivity, we know that determine ABD -> A, ABD -> B, and ABD -> D. We have determined A, B, and D.

Next, we know from transitivity that ABD -> A and A -> BC means that ABD -> BC. Next, we know from decomposition that ABD -> BC means that ABD -> B and ABD -> C. Now, we have determined A, B, C, and D.

We know again from transivitity that ABD -> B and B -> CE means that ABD -> CE. Next, we know from decomposition that ABD -> CE means that ABD -> C and ABD -> E. Now, we have determined A, B, C, D, and E.

Finally, we know from transitivity that ABD -> A, and from augmentation, we know that ABCD -> AC. The former is true since we've determined C from ABD, thus, the FD holds. Next, since AC -> H, we know that ABD -> H. Now, we have determined A, B, C, D, E, and H.

Since we've determined every attribute in R using the axioms, we are done!

4. Compute the canonical cover of F, F_c .

```
Fc = {
     A -> BZ,
     B -> CE,
     D -> B
}
```

5 Question 4

Consider the following relation and set of functional dependencies

```
book(title, author, publisher, location, areaCode)
title, author -> publisher
publisher -> location
location -> areaCode
```

1. Name a candidate key for this relation and show that this is the case using the rules of Armstrong's Axioms.
2. Is this relation in BCNF? Explain your answer.
3. Use the BCNF decomposition algorithm to decompose the relation into a set of relations in BCNF. Show your steps.
4. Is this a lossless-join decomposition? Explain your answer.
5. Is BCNF always dependency-preserving? What about the result of your decomposition? Explain your answer.

5.1 Solution

1. Name a candidate key for this relation and show that this is the case using the rules of Armstrong's Axioms.

{title, author} is a candidate key. First, we know from reflexivity that (title, author -> title) and (title, author -> author). We are given that (title, author -> publisher). We know from transitivity that since (title, author -> publisher), and (publisher -> location), (title, author -> location), so we can determine location. Finally, we apply transitivity again - since (title, author -> location) and (location -> areaCode), (title, author -> areaCode). Thus, since we've shown that we can determine every other attribute in Book using just title and author, {title, author} is a candidate key.

2. Is this relation in BCNF? Explain your answer.

No. Since (publisher -> location) is a functional dependency, but publisher itself is not a superkey for Book, Book is not in BCNF.

3. Use the BCNF decomposition algorithm to decompose the relation into a set of relations in BCNF. Show your steps.

We will iteratively identify dependencies that break BCNF and then split relations up accordingly. First, we notice that (publisher -> location) breaks BCNF, so we split Book into B1(title, author, publisher) and B2(publisher, location, areaCode). B1 has the sole FD of (title, author -> publisher) and B2 has two FDs, namely (publisher -> location) and (location -> areaCode). We notice that B1 is in BCNF and proceed to work on B2.

Next, we identify that (location -> areaCode) breaks BCNF for B2. Thus, we break B2 into two relations, B2'(publisher, location) and B3(location, areaCode). B2' has the sole FD of (publisher -> location) and B3 has the sole FD of (loaction -> areaCode.

Since B1, B2', and B3's FDs are all determined by superkeys, we are done! To reiterate our final result:

B1(title, author, publisher)

B2'(publisher, location)

B3(location, areaCode)

4. Is this a lossless-join decomposition? Explain your answer.

Yes! We know that a title and author uniquely determine a publisher, so this is either a many to one or a one to one relationship. Thus, a join would preserve all of the information between titles, authors, and publishers.

Moreover, since a publisher uniquely determines a location, we have a similar many to one or one to one relationship, which also preserves information between publishers and locations.

The same reasoning follows for locations and areaCodes, and thus, since we haven't broken our relation up into any many to many relationships, we have constructed a lossless join decomposition.

5. Is BCNF dependency-preserving? What about the result of your decomposition? Explain your answer.

Not necessarily. Our result is dependency preserving, since if we compared the set of FDs we have after decomposing from before, we notice that we haven't added or removed any FDs.

However, it is possible to have a BCNF decomposition that doesn't preserve dependencies. Consider the relation R(A, B, C) where A -> B and BC -> A. By BCNF, we would split this into R1(A, B) and R2(B, C). While this is in BCNF, since A -> B and BC -> BC, we have lost BC -> A.

Feedback

As this is a new course, we appreciate any feedback you have for us! If you enjoyed this assignment, hated this assignment, or have other thoughts to share, please do so here!