Warmup #1 (Textbook Problem 8.6)

Answer:

Starting with $A \rightarrow BC$, we can conclude: $A \rightarrow B$ and $A \rightarrow C$.

Since $A \rightarrow B$ and $B \rightarrow D$, $A \rightarrow D$ (decomposition, transitive)

Since $A \rightarrow CD$ and $CD \rightarrow E$, $A \rightarrow E$ (union, decomposition, transitive)

Since $A \rightarrow A$, we have $A \rightarrow ABCDE$ (reflexive), from the above steps (union)

Since $E \rightarrow A$, $E \rightarrow ABCDE$ (transitive)

Since $CD \rightarrow E$, $CD \rightarrow ABCDE$ (transitive)

Since $B \rightarrow D$ and $BC \rightarrow CD$, $BC \rightarrow ABCDE$ (augmentative, transitive)

Also, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow D$, etc.

Therefore, any functional dependency with $A$, $E$, $BC$, or $CD$ on the left hand side of the arrow is in $F^+$, no matter which other attributes appear in the FD. Allow $*$ to represent any set of attributes in $R$, then $F^+$ is $BD \rightarrow B$, $BD \rightarrow D$, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow BD$, $B \rightarrow D$, $B \rightarrow B$, $B \rightarrow BD$, and all FDs of the form $A \star \rightarrow \alpha$, $BC \star \rightarrow \alpha$, $CD \star \rightarrow \alpha$, $E \star \rightarrow \alpha$ where $\alpha$ is any subset of \{A, B, C, D, E\}. The candidate keys are A, BC, CD, and E.
Warmup #2 (Textbook Problem 8.7)

Answer: The given set of FDs $F$ is:

\[
\begin{align*}
A & \rightarrow BC \\
CD & \rightarrow E \\
B & \rightarrow D \\
E & \rightarrow A
\end{align*}
\]

The left side of each FD in $F$ is unique. Also none of the attributes in the left side or right side of any of the FDs is extraneous. Therefore the canonical cover $F_c$ is equal to $F$.

Warmup #3 (Textbook Problem 8.9)

Answer:

a. The query is given below. Its result is non-empty if and only if $b \rightarrow c$ does not hold on $r$.

\[
\begin{align*}
\text{select } b \\
\text{from } r \\
\text{group by } b \\
\text{having count(distinct } c \text{) > 1}
\end{align*}
\]

b. 

\[
\begin{align*}
\text{create assertion } b\text{-to-}c \text{ check} \\
(\text{not exists} \\
(\text{select } b \\
\text{from } r \\
\text{group by } b \\
\text{having count(distinct } c \text{) > 1})
\)
\)
\]
Warmup #4 (Textbook Problem 8.23)

**Answer:** Certain functional dependencies are called trivial functional dependencies because they are satisfied by all relations.

Warmup #5 (Textbook Problem 3.5)

**Answer:**

a. To display the grade for each student:
   
   ```sql
   select student-id,
   (case
     when score < 40 then 'F',
     when score < 60 then 'C',
     when score < 80 then 'B',
     else 'A'
   end) as grade
   from marks
   ```

b. To find the number of students with each grade we use the following query, where grades is the result of the query given as the solution to part a.

   ```sql
   select grade, count(student-id)
   from grades
   group by grade
   ```
Problem 6 (To Be Graded)

1. SELECT license FROM ( 
    SELECT license, COUNT(license) AS accident_count 
    FROM Car INNER JOIN Accident ON Car.license = Accident.license 
    GROUP BY license 
    ORDER BY accident_count DESC) 
LIMIT 5;

2. SELECT a.license, a.accident_date_time 
   FROM Accident AS a INNER JOIN Owns AS o ON a.license = o.license 
   WHERE driver_ssn <> SSN;

3. SELECT p.name, p.street_address 
   FROM Person AS p INNER JOIN Accident AS a ON p.SSN = a.driver_ssn 
   WHERE a.damage_amount > 1000;

4. SELECT SSN ( 
    SELECT DISTINCT p.SSN, SUM(a.damage_amount AS damages) 
    FROM Person AS p INNER JOIN Accident AS a ON p.SSN = a.driver_ssn 
    WHERE p.state <> "RI" 
    GROUP BY p.SSN 
    ORDER BY damages DESC) 
LIMIT 5;

We also accepted:

SELECT DISTINCT p.SSN 
   FROM Person AS p INNER JOIN Accident AS a ON p.SSN = a.driver_ssn 
   WHERE p.state <> "RI" 
ORDER BY a.damage_amount DESC 
LIMIT 5;

Problem 7 (To Be Graded)

1. This brings a functional dependency on customer.zip->customer.region_id.

2. This doesn’t bring a functional dependency; since two customers may have 
   the same address or share a credit card number.

3. You can assume these domains exist or create your own domains, both 
   versions will be accepted.

   • status must be either “Basic” or “Subscriber”.  
     ALTER DOMAIN status ADD CONSTRAINT statchk CHECK (VALUE  
     = ”Basic” OR VALUE = ”Subscriber”);
• price for every item must be non-negative and qty (quantity) of an item in an order must be greater than zero.

ALTER DOMAIN price ADD CONSTRAINT pricechk CHECK (VALUE >0);
ALTER DOMAIN qty ADD CONSTRAINT qtychk CHECK (VALUE >0);

Another acceptable way of doing this is the ALTER TABLE statement:
ALTER TABLE items ADD CONSTRAINT pricechk CHECK (price >0);
ALTER TABLE order_items ADD CONSTRAINT qtychk CHECK (qty >0);

• total and qty in the order must correspond to the total price and total quantity of all items included in the order.

Since ALTER DOMAIN or ALTER TABLE statements cannot access multiple tables, we should consider using a different method. We allowed subqueries in ALTER DOMAIN statements for the sake of this problem. Note: we were very lenient on grading this problem since it was so vague.

Assume value is an order id...

ALTER DOMAIN order_check
ADD CONSTRAINT quantity_check CHECK
  (value not in
    (select o.id
      from (select orders.id as oid, sum(order_items.qty) as total_qty from orders as o, order_items as i where o.id = i.iid) as total_qty where o.id = oid and o.qty != total_qty));

ALTER DOMAIN order_check
ADD CONSTRAINT total_price_check CHECK
  (value not in
    (select o.id
      from (select orders.id as oid, sum(orders.qty * items.price) as total_price from orders as o, order_items as i, items where o.id = i.oid and i.iid = items.id) as total_price where o.id = oid and o.total != total_price));