Solutions

Problem 1

Pet_Name —> Pet_Type is not a functional dependency as people are free to call their pets whatever they want. In fact, “Mr. Ed” is Hayes’ dog and Sherman’s horse and presumably two different pets.

Pet_ID, Visit_date —> Procedure is not a functional dependency either since the same pet can have more than one procedure per visit. For instance, Rover got a Heart Worm Test and had his Wound Treated.

Pet_ID —> Owner, Owner_birthday should be a functional dependency since a pet belongs to one and only one owner. However, if pets can change owner (by purchase or inheritance), then it is *not* a functional dependency.

Pet_ID —> Pet_birthyear is a functional dependency since Pet_ID characterizes a pet uniquely and a pet can have only one birthday.

Pet_ID —> Pet_Name, Pet_birthyear, Pet_Type, Owner_name, Owner_birthday is also a functional dependency if pets do not change owners. It would not be one if pets can change names or owners.

Owner_name, Owner_birthday —> Pet_ID, Pet_Name, Pet_Type. Even though a name and a birthday are often assumed to characterize a person uniquely, a single person can have more than one pet. Indeed, they often do. Just visit my sister.

Problem 2

If we update (correct) any data on a pet or any data on an owner, we have to update these data in all records of procedures involving that pet or involving a pet of that owner. For example, if Rover changes owner because Frederick Douglas dies, or if we want to correct Frederick Douglas’s birthday, we need to do so at peace three times. Similarly, if it turns out that Rover’s treatment for a wound was on a different rate, then we would have to correct two different records.

If we delete some procedure records, we can loose information about a certain pet or an owner. For example, there is only one record about Mr. Ed (the dog) and if we decide that this procedure was actually not done and remove the record, then we loose Hayes’ birthday.

Problem 3

$r_1(x)r_2(x)w_1(x)w_2(x)$ looses the update of Transaction 1 as Transaction 2 reads the previous value of x and updates based on it. Thus, the effects of Transaction 1 are lost.

$r_1(x)w_1(x)r_2(x)w_2(x)$ is fine. Transaction 1 operates before Transaction 2.

$r_1(x)r_2(x)w_2(x)w_1(x)$ looses the update of Transaction 2.

$r_2(x)w_2(x)r_1(x)w_1(x)$ is fine. Transaction 2 is performed before Transaction 1.

Problem 4

$r_1(x)w_1(x)r_2(x)r_1(y)w_1(y)r_2(y)$ is fine. Transaction 2 always reads the results written by Transaction 1. In fact, this is a serializable schedule.

$r_2(x)r_1(x)w_1(x)r_1(y)w_1(y)r_2(y)$ has a potentially inconsistent read as Transaction 2 reads the x-value before Transaction 1 writes it and the y-value after Transaction 1.

$r_2(x)r_1(x)r_2(y)w_1(x)r_1(y)w_1(y)$ is fine. Transaction 2 always reads the result before Transaction 1 changes them. The schedule is serializable and corresponds to a schedule where Transaction 2 precedes Transaction 1.

$r_2(x)r_1(x)w_1(x)r_2(y)r_1(y)w_1(y)$ is also fine for the same reason.

Problem 5

Given
$r_2(x)w_2(x)r_3(x)w_3(x)r_2(y)r_1(z)w_1(x)w_2(y),$
we transpose $w_1(x)$ and $w_2(y)$. This gives
$r_2(x)w_2(x)r_3(x)w_3(x)r_2(y)r_1(z)w_2(y)w_1(x).$
We then transpose $r_1(z)$ and $w_2(y)$. The result is
$r_2(x)w_2(x)r_3(x)w_3(x)r_2(y)w_2(y)r_1(z)w_1(x).$
At this point, Transaction 1 is after the other two transactions. We now switch $r_2(y)w_2(y)$ first with $w_3(x)$ (two switch operations) and then with $r_3(x)$ (also two switch operations) to obtain
$r_2(x)w_2(x)r_2(y)w_2(y)r_3(x)w_3(x)r_1(z)w_1(x),$
which is a nice serial schedule.