1.
a) You need to determine what constraints you want to impose to prevent the database from becoming inconsistent and incomplete. To achieve this you can apply several constraints such as enforcing that some attributes always contain a value, that they are not allowed to be nulls. You can also establish the domains of attributes, the set of values an attribute can take. There are also constraints regarding integrity; a primary key can't be null. As for referential integrity it is important to consider different possibilities of behavior as in delete. In some occasions it is better to set null on delete, or cascade.

b) The referential integrity constraints can be to Set Null on Delete of a tuple from a relation. This way the foreign key will have a null value and reflect the state where the scientists haven't decided what family a genus belongs to.

2.
The first scenario should use an aggregation. The reason is that the question specifies that the application is primarily for customer service. If we use a composition, the customer will not be allowed to exist till she has placed an order and we won't be able to register customers into our database till they have placed orders, which is a bad marketing strategy.
The second scenario should use a composition. Law enforcement is not interested in who we're targeting in our ads/promotions or who we've got in our database for customer service reasons. They are only interested in looking up customers who have actually bought medicines so they can check for prescription and drug abuse. For this reason, composition is a natural fit for this problem.

3.
For the solution in the lab as we have shown it, the change would consist of adding 3 attributes to the association class Road Segment. These would be From_Direction, To_Direction and isOneWay. The first two of these are integers and are foreign keys to the Intersection class, thereby identifying the direction in a precise manner. The third is a boolean that checks whether the road segment is only one way. This may seem redundant and may be neglected but it helps to speed lookup and helps in consistency maintenance(see below).
One thing to note here is that if isOneWay is false, there will be a corresponding instance of Road Segment in the other direction with the From and To reversed. isOneWay gives us a way of checking that this always holds (maybe through a trigger or constraint) so that the database is in a consistent state.

Common mistake: Many people had similar answers to the one above except they were liberal with NULL values for certain attributes and relied on an 'application' to check for this or that. Both of these are *usually* bad ideas. Databases are often serviced by multiple apps that make use of the same database (making concurrency a big headache as well, which is why the whole concept of 'transactions' exists) so we don't want to rely on them to make strong assumptions. NULL values should be avoided where possible as 'placeholders' for something that might not exist because it can also stand for 'unknown'. What if I don't know whether the road segment is one way or two way? I might try placing a NULL there. But if the database designer has decided that a NULL for an attribute 'implies' something else, then my NULL could end up having serious consequences.