Question 1
As per the questions, the default statement is lions are dangerous. And a strong statement, that the baby lions are not. Provided, that Helios is a baby lion. This is a complete information as per the question. My program makes sure that if I add any other baby lion in the program, it shows, that it's not dangerous. For testing purpose, I added a baby lion named Rogers, it's accepting the flow. For default statement, my code is -   dangerous(X) :- lion(X), not ab(d(X)), not -dangerous(X).

My output shows:

```
dangerous(apollo) -dangerous(helios) -dangerous(rogers)
```

Question 2

Given in question, default statement is married people like each other. With the strong statement, Bob hates Mary. The default code is: likes(X,Y) :- married(X,Y), not ab(d(X)), not -likes(X,Y). This can be extend to the exception for default. *I also used the logic that, if X is married to Y then Y is also married to X.* Similarly, the hate goes as , -likes(X,Y) :- hates(X,Y).

A) My solution answers, - (this solution answered to all given questions in this part, see below)
B) The provided solution below, shows that since the Kate behavior is unpredictable hence the default condition applied on kate, hence the predicate likes(Kate, Arnold) is unknown. And it answers yes to likes(Arnold, Kate).

```
hates(bob,mary) likes(john,susan) likes(arnold,kate) likes(mary,bob) likes(susan,john)
```

Question 3

As given in the question, there are two default statements i.e. car normally have four seats and pickups normally have two seats. Provided exceptions are pickups and extended cabs. I designed the code as: four_seats(X) :- car(X), not ab(d(X)), not -four_seats(X) and two_seats(X) :- pickup(X), not ab(d(X)), not -two_seats(X). Then I provided a condition that I assumed that two seats are not four seats. My clingo solution gives the answer:

```
four_seats(c) four_seats(a) four_seats(d) two_seats(b)
```

Question 4
This question was pretty straightforward. It says the condition of can_graduates, i.e. if student has completed all the courses. For this, I differentiated between graduates and -graduates as -can_graduate(X) :- student(X), not can_graduate(X), because I can show my solution sets transparently.
Provided the knowledge base and the code, my solution leads to as per clingo:

```
can_graduate(john) -can_graduate(mary)
```

Question 5

This is the follow-up question of question 4. As given in the question the list is incomplete. As given in the question, I focused few points i.e. *Bob information of taking graph and Rick complete information was missing.* Though given that Bob did not took math class. Following this incomplete information, I formed a default in students who can graduate as can_graduate(X) :- student(X), not ab(d(X)), not -can_graduate(X). Because, may be the case that Bob had taken the graph class and Rick is having no information. And the exception to this are the student assumed who not took classes i.e. ab(d(X)) :- student(X),course(Y), not took(X,Y).
Then I designed the logic for students who can't graduate i.e. -can_graduate(X) :- student(X), course(Y), -took(X,Y). This means students who missed any of the course cannot graduate, provided by question 4 information. This logic clears the fact of Bob as provided Bob did not attend the math class, hence he can't graduate despite the fact he had or had not taken maths class.
Then I designed the review records as, review_records(X) :- student(X), not can_graduate(X), not -can_graduate(X), this tells me that students who are missing under graduate and not graduate column due to missing information should rely in review_records. My solution in clingo gives the following:-

```
can_graduate(john) review_records(rick) -can_graduate(mary) -can_graduate(bob)
```

Question 6
Given in the question, I made two classes of saxophone and Selmer Mark VI and this is also a subclass of saxophone. As per provided information, every person saxophone is known by its name, hence I used is_a(jake, selmerMarkVI) type relation to show the naming of saxophone. Again, as per provided information of high Dkey and spring, I used part_of(d_key, X):- member(X, saxophone) & part_of(spring, X):- part_of(d_key, X) for showing the relation between them along with saxophone i.e. if highDkey belongs to X then X belongs o saxophone or saxophone name is X. Same with spring as if spring belongs to X then spring related highDkey also belongs to X.
Then I worked on the default, normally springs do not broke is designed as -broken(X,spring):- member(X,saxophone), not ab(d(X)), not broken(X,spring). Here, I'm giving the default of spring not broke. Along, provided knowledge base, I used broken(mo,spring), as Mo has broken spring. Then the predicate used for broken saxophone is used as: broken_sax(X):- member(X, saxophone), broken(X,spring) and the for not broken: -broken_sax(X):- member(X, saxophone), not broken_sax(X). From this the solution set will represent all the relation along with the answer of person having broken saxophone and not broken saxophone. My solution for given question in clingo:

```
broken_sax(mo) -broken_sax(jake)
```