For #1–4, the problem stated that "wrong" means "can't store some valid data", and"worse" means "can store all valid data but has drawbacks". This can be a bit of gray area when it comes to cardinality and participation constraints. With cardinality, allowing N instead of 1 doesn't prevent storing valid data but it is still wrong — the perpective is that it reflects a cardinality of N rather than 1 rather than just not enforcing the expected cardinality of 1. On the other hand, not enforcing total participation is more of a drawback — "worse" rather than "wrong". The label (worse vs wrong) was circled if you picked one other than what was intended, but what matters most is identifying the issue(s) with the mapping and being able to weigh the importance of those issues when it comes to choosing between alternatvies.

The relational schema should capture what the ER diagram expresses but it doesn't necessarily have to do it in the same way as the ER diagram.

- For example, in #1, `vehicle_id` being a PK in the diagram means that there is only one license plate value associated with a given vehicle ID. The unique constraint for `vehicle_id` in #1a captures the first part — only one row of `OPERATES` with a given vehicle ID means there can't be more than one license plate for that vehicle. An attribute doesn't necessarily have to be a PK in the relational schema just because it was a PK in the ER diagram. (The other aspect of being a PK — that `vehicle_id` is required for a vehicle i.e. you can't have a license plate value without a vehicle ID — isn't captured in #1a. This can't be captured with the single relation approach — making `vehicle_id NOT NULL` means you can't have a library without a bookmobile.)
- Similarly, there doesn't need to be a relation in the schema for every relationship in the ER diagram — in #1b, the connection between the branch and the bookmobile that `OPERATES` represents is captured by the FK `LIBRARY_BRANCH.vehicle_id` → `BOOKMOBILE.vehicle_id`.

It's not enough for the relational schema to simply capture a relationship — it should also capture the cardinality and participation constraints. So, for #1b, it's not enough to just have the FK — `branch_num` as PK and unique for `vehicle_id` captures the 1:1 cardinality and the not null aspect of `branch_num` as PK captures the total participation of `BOOKMOBILE`. (Allowing null for `vehicle_id` reflects the partial participation of `LIBRARY_BRANCH`.)

An attribute being a PK in one relation means not null only for that table. For example, in #3b, `BOOK.isbn` being PK means that `BOOK.isbn` is not null but it doesn't preclude `WROTE.isbn` from being not null — that's why a not null constraint for `WROTE.isbn` is explicitly stated. (Also, the referring end of an FK can be null — in #1d, a book's author can be null even though `author_id` is an FK.

#3e — `(author_id,isbn)` as a composite PK allows the M:N cardinality of `WROTE`, but leads to repeated information when a book has mutiple authors or an author writes multiple books and, more critically, doesn't allow for books without authors (allowed

by the ER diagram because `BOOK` has only partial participation in `WROTE`).

When a PK is composite, as with `AIRPLANE_TYPE` in #6, the FK referring to it must also be composite (not two separate FKs). `(AIRPLANE.maker,AIRPLANE.model)` → `(AIRPLANE_TYPE.maker,AIRPLANE_TYPE.model)` means that an airplane's particular combination of maker and model must exist as an airplane type; stating `AIRPLANE.maker` → `AIRPLANE_TYPE.maker` and `AIRPLANE.model` → `AIRPLANE_TYPE.model` separately means there needs to be an airplane type with that maker and one with that model, but doesn't require that those be the entities.

PKs and FKs are part of the relational schema along with the relations and their attributes — include PKs and FKs!

For #5, give the schema for the whole diagram, not just the specialization part. (It is OK to factor out the common parts and not repeat what is the same for each part of the problem, but don't omit them entirely from your answer.)

For #6, `FLIGHT_LEG` either needs a partial key of its own (such as `leg_num`) or `ARRIVES_AT` and `DEPARTS_FROM` to be identifying relationships — otherwise there can be only one flight leg per flight. (Why?) The latter was intended (given the total participation of `FLIGHT_LEG` in those relationships and the N:1 cardinality), but any reasonable approach to fixing a problem in a problem is fine *as long as you state that is what you are doing*. Otherwise it looks like a mistake or that a solution was copied from somewhere.