Responses

Task 1

- *Discuss why it is important to review the final model. Consider the cases to make changes in the later stages of database development.*

The conceptual model describes and identifies the high-level abstraction of data objects without delving into any specifics of a database system (Elmasri & Navathe, 2016). It is thus important to review the final model so that it is ensured that it meets several criteria:

1. Simplicity—it should be easy to understand what the model describes so that mistakes during the system development process are mitigated early on
2. Unambiguous—there should be no guessing as to what entities the model describes and their relationships. The model would help database designers to create system-specific schemas that address the domain. So, the conceptual model should not lead them into erroneous design decisions
3. Correctness—the semantics and syntax of the model should be precise and accurate so that the designers can capture the intended concepts, entities, and relationships that address the domain

Without these and others; such as completeness, minimality, and readability; it is highly likely that the subsequent development processes would be problematic at best.

- *There are several issues to consider when reviewing the final model. One of the issues is `M:N` relationships. `M:N` relationships should be removed while reviewing the final model. Discuss the problems of `M:N` relationships and how they should be resolved. Support your discussion with examples.*

Whereas the `M:N` relationship is one of the possible cardinality ratios for binary relationships among entities, it poses issues in the final model (Elmasri & Navathe, 2016):

1. It makes the resultant database tables have complex relational operations—and these are likely to lead to errors in outputs and systems inefficiencies
2. It generates numerous redundancies in the resultant database tables

These issues are evident in a case where we have `Student` and `Class` entities, for instance.

A `Student` could take many classes. In addition, a `Class` may have many students.

Thus a `Student` table could look like:

> *Primary key:* ***student_id*** + ***class_id***
>
> *Foreign key:* ***class_id***

student_id	name	class_id
123	Fred	100
123	Fred	110
456	Jane	100
456	Jane	110

Where the `Class` table is:

> *Primary key:* ***class_id*** + ***student_id***
>
> *Foreign key:* ***student_id***

class_id	class_location	student_id

class_id	class_location	student_id
100	West	123
100	West	456
110	East	123
110	East	456

These tables display many redundancies. The `student_id` column in the `Student` table, for instance, contains several rows with matching values. The same is true for `class_id` in the `Class` table also.

These issues can be resolved by introducing a bridging table. One that would contain the repeated values from both `Student` and `Class` tables.

Such a table could be named `Enroll`. Moreover, it would turn the `M:N` relationship int one of `1:N`:

> *Primary key:* ***student_id + class_id***
>
> *Foreign key:* ***student_id, class_id***

student_id	class_id
123	100
123	110
456	100
456	110

Whereas the table `Enroll` seems overly similar to the former `Student` table. The latest `Student` table now looks like:

student_id	student_name
123	Fred
456	Jane

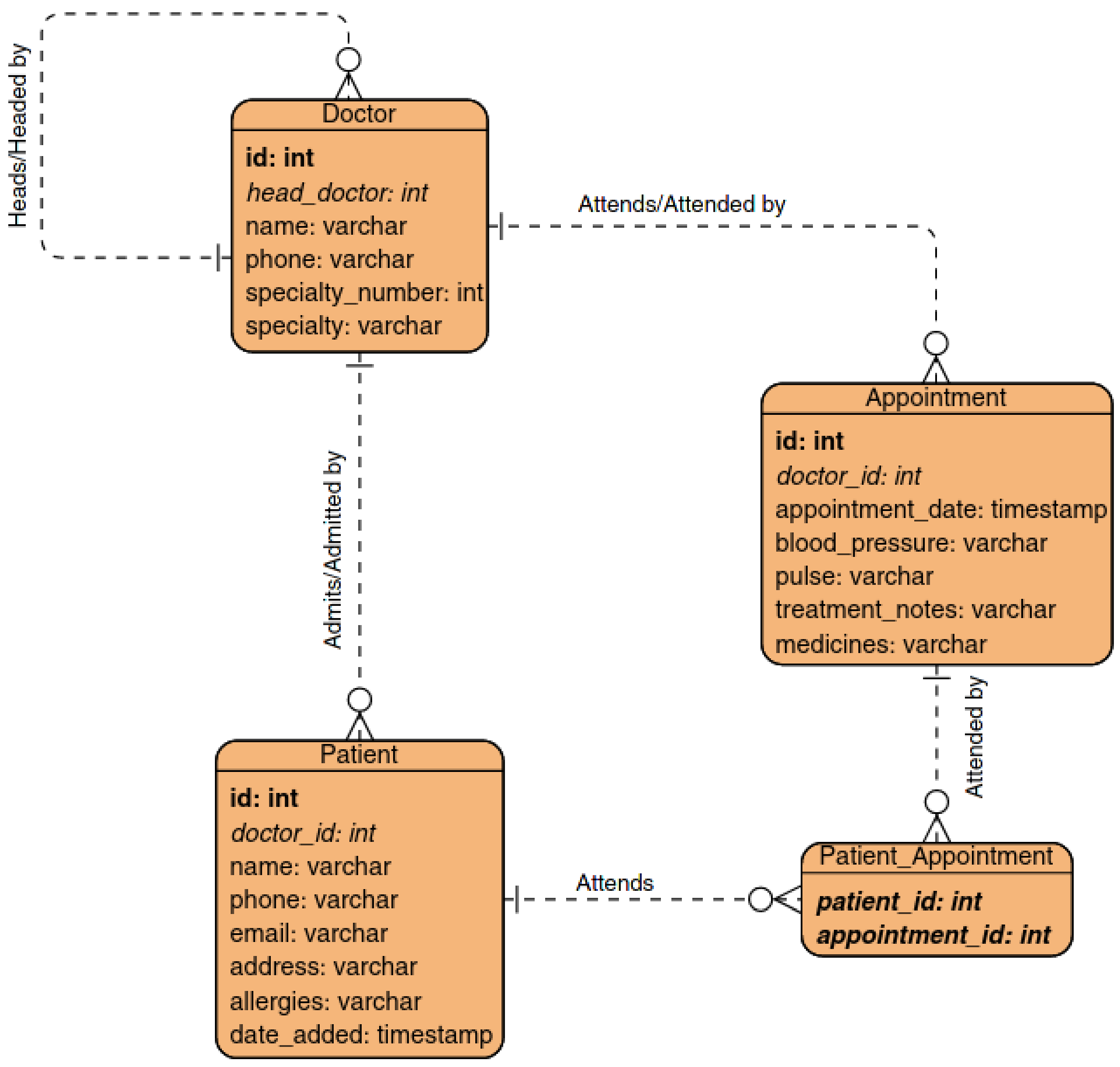
And the new `Class` table:

class_id	class_location
100	West
110	East

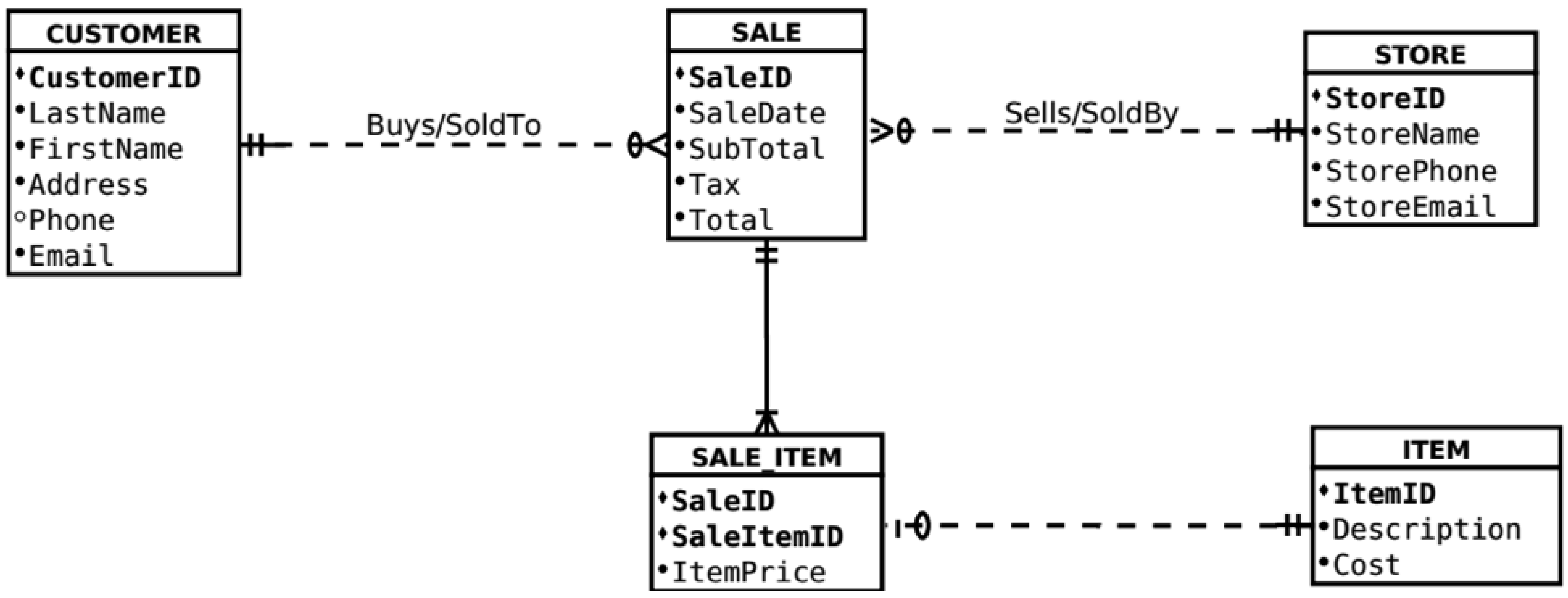
Reference

Elmasri, R., & Navathe, S. (2016). *Fundamentals of database systems*. Pearson.

Task 2



Task 3



Write a scenario where the user described requirements during an interview. Requirements described in this interview should be transformed into the provided E-R diagram. After writing the scenario, you should also describe the cardinalities in the provided diagram.

A customer hears of a sale that's ongoing in a local store.

The event will offer several items up for sale.

Because the customer needs to be alerted whenever such sales occur, she has decided to deposit her contact details at the store so that she can get timely alerts. In turn, the customer has saved the contact details of the store so that she can call up or write an email in good time for any inquiries or complaints she may have.

On closer inspection, she has realized that whenever an item comes up for sale, it is smart to wait for a few days before going into the store to purchase that item.

The items on sale seem to be priced lower and lower with each passing day as the sale progresses.

On the other hand, the customer lives in a city where putting items up for sale is a regular occurrence for many of the local stores.

These stores usually put up the sales when the end of a season for a particular item nears or ends.

So the customer has many sales to choose from at a given time.

Yet, what that customer does not realize is that many other customers get interested in those sales just like her. So a given store is likely to host many customers in a day when a sale is ongoing.

This scenario thus suggests that there are four entities that interact during sale events. They include: `customer`, `store`, `sale`, `sale item`, and `item`.

The distinction between `item` and `sale item` is important because nit all items go on sale during a sale event.

Nonetheless, a `customer` has information such as a `unique identifier`, `last name`, `first name`, `address`, `phone number`, and `email address`.

A `store` has information such as a `unique identifier`, `store name` `phone number`, and `email address`.

The `store` stocks `item` entities that have information such as a `unique identifier`, `description`, and `cost`.

Whenever the `store` puts up a `sale`, it ensures it has information such as a `unique identifier`, `date`, `sub total`, `total sales`, and `tax`.

Because the `store` is the entity that puts up the `sale`, then it shows that they have a relation. And such a relation is one where a `store` puts up many sales in a given period of time. So the cardinal relation between the `store` and the `sale` is one-to-many, `1:N`.

Also, `customer` entities take part in many `sale` events, meaning that they have a relation too. One customer can attend many `sale` events. That means that the relation between a `customer` and a `sale` is one-to-many, `1:N` also.

Additionally, specific `item` entities are put for sale—not all of them. Thus, there should be a distinct entity named `sale item`. That entity should information such as a `unique identifier` and `sale price`. Still, that `sale item` should have a relation with a conventional entity named `item` that may or may not be on sale. And the `item` should have information such as a `unique identifier`, `description`, and `cost`.

Otherwise, a sale event offers many `sale item` entities. That means that a `sale` and a `sale item` have a `1:N` relationship.

Also, an `item` can be offered in many sale events; meaning that the relation between an `item` and a `sale item` is of the type `N:1`.