There are #3 ways we can map inheritance relationship between the entity classes into relational model
1. SINGLE_TABLE
Representing all the entity classes in the hierarchy, create only one single database table. To determine which record belongs to which entity class in the hierarchy and to support polymorphic access we need to use discriminator column

drawbacks:
1. de-normalized tables/schema, that results in huge wastage of memory. Lot of columns are left with NULL per each record while storing the objects in the hierarchy into one table

advantages:
1. no table joins that results in max performance

2. JOINED
Representing the entity classes in the hierarchy we create tables with joined relationship with the superclass table. For the super class create a separate table, and per each subclass create a table, joined relationship with the superclass table

drawbacks:
1. during the time of persisting (insert/update/delete) the subclass objects, we need to execute #2 queries in storing the data which requires more I/O on the database that impacts performance
2. while querying the data for a subclass, we need to execute an joined query with the superclass table, which degrades the performance
3. the worst part is, while querying the data for an parent class object we need to perform an left outer join across all the tables in the hierarchy to support polymorphic access that results in poor performance and not suitable for large volumns of data or more classes in the hierarchy.

advantages:
1. fully-normalized tables, results in optimized storage (without any memory wastage)

3. TABLE_PER_CLASS
Representing the entity classes in the hierarchy we need to create separate tables per each class, which is independent of each other

drawbacks:
1. The superclass columns are duplicated or repeated across all the subclass tables in the hierarchy, that results in maintainability problem. Any change in the superclass attributes not only affects the superclass table, all the subclass tables also needs to be modified

2. To support polymorphic retrieval, we need to perform union query across all the tables in the hierarchy, that results in poor performance

3. Not normalized representation

advantages:
1. no joins, no memory wastage even though not normalized.