Single table inheritance is a way to emulate object-oriented inheritance in a relational database. When mapping from a database table to an object, a field in the database identifies what class in the hierarchy the object belongs to. (Wikipedia)

STI is an instance of right tool for the right problem

Single Table Inheritance allows you to create new classes that have methods specific for that class/ that aren't shared by the other classes, but inherit the data attributes and maybe some basic behavior from the parent model.

Ask yourself

- **Are the objects, conceptually, children of a single parent?**
- **Do you need to do database queries on all objects together?**
    - If you want to list the objects together or run aggregate queries on all of the data, you'll probably want everything in the same database table for speed and simplicity, especially if there is a lot of data.
- **Do the objects have similar data but different behavior?**
- **Thank you to Alex Reisner for these useful questions, also found here

- Caution: Be careful to not overuse this tool.
    - Ask yourself: is the DRYness that maintained by using single table inheritance worth the added dependency that you are creating. Object oriented design prefers decreasing dependencies whenever possible since as dependencies grow more complex, the code becomes brittle and more costly to change since changing one class could have repercussions in classes that are dependent on it.

How to do it:

1. Run a migration
    a. Create a new column with a "type" data type
2. Separate the model into different classes, and have them inherit from the original class
3. Change any logic necessary in the controller to route the behavior to the new classes

Resources:
http://eewang.github.io/blog/2013/03/12/how-and-when-to-use-single-table-inheritance-in-rails/
http://steinkamp.us/502/alternative-single-table-inheritance-for-rails
http://www.alexreisner.com/code/single-table-inheritance-in-rails

Notes:
Using a single table to reflect multiple models that inherit from a base model, which itself inherits from ActiveRecord::Base.

Useful:

- For:
  - Creating sub-types
    - Guest Users and Logged in Users
    - Different types of blog posts
- When:
  - Attributes are the same but behavior differs

-

Cons:

- Parent table must have columns for attributes of all children models
  - So: Lots of un-used columns for models that don't hold those unique attributes
- **It prevents you from efficiently indexing your data.** Every index has to reference the type column, and you end up with indexes that are only relevant for a certain type.

Alternatives:

If there aren't enough shared data attributes to warrant STI and what is often repeated are similar methods, then the correct answer could be using a module and injecting it into the various classes