Consider the following scenario and assume you have been asked to design and create three relations (museum, artifact, and collection) to represent the corresponding information.

- Museums manage collections. Each collection has to be managed by exactly one museum.
- Collections consist of artifacts. Every artifact belongs to at most one collection (but some are not currently part of a collection).
- Each museum has a unique name (e.g., “Louvre”). Museums are located within a city and a country. The city and country need to be stored as separate attributes (so each can easily be searched independently). Each museum has a curator that is identified by their first and last name (again, as separate attributes).
- Each collection has a unique collection number. Collections have optional names (which do not have to be unique) as well as optional descriptions (which are in plain text).
- Artifacts have unique reference numbers. Reference numbers start at 1000 (i.e., reference numbers below 1000 are invalid). Each artifact has a type, which is either “painting”, “drawing”, “sculpture”, or “other”. Artifacts have creation dates as well as acquisition dates. Note that the acquisition date cannot be before the creation date. For artifacts that are part of collections, the date the artifact joined the collection must be stored. Note the collection date must be after the acquisition date.

1. Write an SQL CREATE TABLE statement for the museum relation. Be sure to include appropriate data types, NOT NULL constraints, primary keys, foreign keys, unique constraints, and any other relevant attribute- and/or table-level constraints.

2. Write an SQL CREATE TABLE statement for the collection relation. Be sure to include appropriate data types, NOT NULL constraints, primary keys, foreign keys, unique constraints, and any other relevant attribute- and/or table-level constraints.

3. Write an SQL CREATE TABLE statement for the artifact relation. Be sure to include appropriate data types, NOT NULL constraints, primary keys, foreign keys, unique constraints, and any other relevant attribute- and/or table-level constraints.

Use the relations below (assuming appropriate data types and constraints) for the following questions. Assume the tables are part of a large conglomerate that owns multiple companies.
company(company_name, industry, date_formed, company_type)

- Note that only the formation date (the date the company was incorporated) and the company industry are required (other than the company name). The industry attribute states the business the company is in (e.g., “software”, “manufacturing”, and so on). The company type states whether the company is a nonprofit, for-profit, educational organization, voluntary organization, and so on.

employee(emp_id, emp_name, works_for, hire_date, title, age)

- Note that works for is a required attribute and is a foreign key to the company table (in other words, all employees work for exactly one company). The employee’s name, hire date, title, and age are also required.

manages(manager_id, emp_id, project_name)

- Note that both the manager id and employee id are foreign keys to the employee table. Note also that the project name (i.e., the project that the manager manages the employee on) is optional.

4. Write an SQL query to find the names of all of the non-profit software companies.

5. Write an SQL query to find the names and hire dates of all employees with the title “software developer” that are between 25 and 30 years old (inclusive). Your query should return the employee names sorted alphabetically.

6. Write an SQL query to find the employees of software companies with the title “software developer.” Your query should return the company name, the company industry, the employee name, and the employee age. In addition, the results should be sorted alphabetically by company name followed by employee name. Your query must use the “JOIN” syntax for all joins.

7. Write an SQL query to find the employees that are under 40 that manage at least one other employee. Your query should return each matching employee’s id and name exactly once. The results should be sorted by employee id (from smallest to largest). Your query must use the “JOIN” syntax for all joins.

8. Write an SQL query to find the names of all companies and their corresponding industries that have at least one employee whose title is “senior software developer” that manages at least one employee whose title is “software developer”. Your query should only return each matching company once (i.e., only unique companies). Your query must use the “JOIN” syntax for all joins.
9. Write an SQL query to find the id's and names of managers that manage at least one employee outside of a project (i.e., where the project has the value NULL). Your query should only return each such manager once. Your query must use the "JOIN" syntax for all joins.

10. Write an SQL query to find the managers $e_1$ that manage at least one employee $e_2$ who is also a manager of at least one employee $e_3$. Your query should return the id and name of manager $e_1$ and the id and name of employee $e_3$. Your query must use the "JOIN" syntax for all joins.