# CECS 630, Spring 2024
# Take-home 1

Antonio Badia

**Due Date: A week from today**

For this takehome, you must use PostgreSQL. For help with Postgres, you can use the class notes, the (excellent) online documentation, or use the Web (`stackoverflow.com` is highly recommended, but you can use whatever you want). For questions about the takehome, you should ask the instructor.

**What to turn in:** an SQL script containing all SQL statements (Create Type/Table, Insert Into, Select, plus any additional statement you may need) used to complete the assignment. Send the script as an email attachment with Subject "630 Takehome 1". Note: this script will be run in a PostgreSQL server to make sure it's correct. You must ensure that the script you submit can execute without problems, as a self-contained unit. You should start by creating a database, connecting to it and creating all schemas within this database.

I have written an app with the following classes (pseudo-code):

```
class Publisher{
 String name;
 String address;
 String country;
 String CEO;
 Set<String> bookspublished;
};

class Author {
 int id;
 String name;
 int age;
 String country;
 Set<String> bookswritten;
};

class AuthorWithAgent extends Author {
 String agentname;
}

class Book {
 String isbn;
 String title;
 double price;
 int year;
 String publishername;
 Set<Integer> authors;
};
```

Note the following:

- You can assume isbn is a key on Book, id is a key on Author, and name a key in Publisher.
- Attribute 'bookswritten' in Author refers to sets of isbns in Book, and attribute 'authors' in Book refers to sets of ids in Author. That is, this reflects a many-to-many relationship between Author and Book.
- In Book, attribute 'publishername' refers to name in Publisher; attribute 'bookspublished' in Publisher refers to sets of isbns in Book. That is, this reflects a one-to-many relationship between Publisher and Book.
- Class AuthorWithAgent inherits from class Author.

Assume the app has large collections of publishers, books and authors, and needs to save this to a database. I'm trying to decide how to design this database.

1. To help me design the database, do the following:
   (a) create a schema for the data in a plain relational database (that is, each attribute in each table should be of a basic data type; no complex types, no collections). All the tables should in 3NF. Make sure to declare all primary keys and all foreign keys.
   (b) After creating this database, insert at least 2 tuples for Publisher, at least 4 for Book, and at least 3 for Author. You can make up this data, but make sure the tuples are all connected.
   (c) create a schema for this data in an extended relational database, using only one (typed) table (you can have several types, and collections, but all data should go in one table).
   (d) Insert the same data that you made up for the relational database in this new schema. NOTE: you can insert this data using a query over the previous database, or manually.
   (e) Create a table with one column of type XMLType **or** of type json (or jsonb) -your choice.
   (f) Insert the same data from the previous databases in this new schema, i.e. transform your data to XML (or JSON). NOTE: you can insert this data using a query over a previous database, or manually (bonus points if you use a query).
2. To help me decide which database is better, run the following queries (representative of my workload) in each database (the plain relational one, the extended relational one, and the one with an XMLType or JSON):
   (a) list, for each author, his/her name, the average price of his/her books, and (if they have one) his/her agent name.
   (b) list, for each book, its title and number of authors (but count only authors younger then 25).
   (c) list, for each publisher, its name and the title of books it published in 2015.
   (d) list names of authors who have published a book together (i.e. pairs (a,b) such that 'a' and 'b' have written a book together). Note that the book may have more than two authors; also, 'a' and 'b' may have written several books together.

   Note: on the plain relational and extended relational databases, the answer should be a table. On the XMLType database, your answer should be in XML; on JSON, it should be JSON.

**Open Questions** Answer *one* of the following. NOTE: if you need extra time for this part, you can ask for it as far as you submit the previous part on time.

1. As we've mentioned in class, M-N relationships, when expressed in XML, result in redundancy. A similar problem happens when we use subtypes and embedding. Hence, the databases you designed above contain redundancy. Design an alternative (extended relational) database that stores the data above in as few tables as possible, but eliminates any redundancy among Books or Authors due to their M-N relationship. Redo all queries above on this new database.

2. Give a procedure to store an XML dataset in a single database table by using structured types and collections to capture the structure of the XML data. That is, adapt the procedure shown in class for 'shredding' XML data into tables (which assumed plain relational tables) to a database with types, arrays and multisets. Your input is a DTD for the XML dataset; your output is a database schema (including all necessary types). Then, apply your procedure to the following example:

   ```
   (author (name age address* books+))
   (address (number street city zip))
   (book (isbn title year publisher))
   ```

   Assume that any element that is not further described (e.g. `name, year`) is an atomic value (no parts).

3. Assume the following tables, similar to an example shown in the slides:

   ```
   create type lineitem as (
   partKey int,
   suppKey int,
   unitPrice numeric,
   quantity int);

   create table Myorder(
      orderKey int primary key,
      custKey int,
      orderstatus int,
      totalPrice numeric,
      orderDate date,
      items lineitem[]);
   ```

   Write the following queries:

   (a) List the order key and order status for orders where at least two lineitems have quantities greater than 10.

   (b) List the order key and order status for orders where some lineitem has the same supplier (suppKey) as another lineitems (i.e. there is some supplier that supplies two of the lineitems).

   (c) List, for each order, the average number of lineitems (quantity).

   (d) The *cost* of a lineitem is its unit price times its quantity; the cost of an order is the sum of the costs of all its lineitems. Find all orders where their cost is different from their total price.

   Note that, since your tables are empty, the above queries will return empty answers; however, you should write them thinking about a database with a large number of orders, each with (up to) dozens of lineitems.