

**CSC 455: Database Processing for Large-Scale Analytics****Assignment 1****Due: 11:00 pm, Friday, October 2<sup>nd</sup>.**

**Supplemental reading:** SQL reference book *Oracle 11g SQL* by Price, ISBN 9780071498500 (available in Books 24x7 DePaul online library as eBook). Sections 1.1, 1.2, 1.4, 1.6

<http://library.books24x7.com.ezproxy.depaul.edu/toc.aspx?site=XOBDU&bookid=20722>

**Part 1**

Create a set of relational schemas with underlined (primary) keys and arrows connecting foreign keys and primary keys for a database containing the following information. If you have any difficulty drawing arrows, you can write foreign key information in a sentence instead.


- **Authors** have Last Name, Firstname, ID, and Birthdate (identified by ID)
- **Publishers** have Name, ID, address (identified by ID)
- **Books** have ISBN, Title, Publisher (each book has a unique publisher and can be identified by ISBN).
- Authors **Write** Books; since many authors can co-author a book, we need to know the rank of an author contributing to a book, stored in this table (i.e. a number 1, 2, 3; for single author books, this number is 1).

NOTE: Part 2 has some sample data which may be helpful.

Author(Author\_ID, Author\_Lastname, Author\_Firstname, Author\_birthday)

Publisher(Publisher\_ID, Publisher\_Name, Publisher\_address)

Book(ISBN, title, *Publisher\_Name*)



[Note: Underline = PRIMARY KEY; Italic = FOREIGN KEY]

The Primary keys for the relational schemas are: Author\_ID, Publisher\_ID, and ISBN. They link/dictate and correspond to the other column information.

The foreign keys originate with the Books schemas (publisher) -> Publisher\_Name -> Publisher\_ID -> Author\_ID. We can assume that authors have only one unique publisher.

## Part 2

- Using your logical schema from Part1, write the necessary SQL DDL script to create the tables. Be sure to specify every primary key and every foreign key. You can make reasonable assumptions regarding the attribute domains.

Code File attached: assignment1\_sql\_ddl.sql

- Write SQL INSERT statements to populate your database with the following data (NOTE: remember that strings would need to use single quotes, e.g., 'Asimov')

Code File attached: assignment1\_sql\_insert.sql

- Write a python function that is going to generate and return a SQL INSERT statement given a table name and value list as parameters. For example, generateInsert('Students', ['1', 'Jane', 'A-']) should return “INSERT INTO Students VALUES (1, Jane, A-);”. It would be even better, but not required if your function returned the more proper “INSERT INTO Students VALUES (1, 'Jane', 'A-');” (i.e., put quotes around strings, but not numbers). Another example: generateInsert('Phones', ['42', '312-555-1212']) would produce “INSERT INTO Phones VALUES (42, 312-555-1212);” For simplicity, you can assume that every entry in the list of values is given as a string, even if it is a number.

```
chars = []

def generateInsert(tablename, lst):
    for entry in lst:
        if entry[0:] <= '9' and entry[0:] > '-': # if its a number and not a symbol by values of letters and symbols
            val = entry
        else:
            chars.append("{}'".format(entry)) # all letters/grades

    sqlInsert = "INSERT INTO {} VALUES ({} , {} , {});".format(tablename, val, chars[0], chars[1])
    return sqlInsert

print(generateInsert('Students', ['1', 'Jane', 'A-']))
```

Code File Attached: generateInsert.py

### Part 3

Consider a MEETING table that records information about meetings between clients and executives in the company. Each record contains the names of the client and the executive's name as well as the office number, floor and the building. Finally, each record contains the city that the building is in and the date of the meeting. The table is in First Normal Form and the primary key is (Client, Office). (Date, Client, Office, Floor, Building, City, Executive)

You are given the following functional dependencies:

Building  $\rightarrow$  City

Office  $\rightarrow$  Floor, Building, City

Client  $\rightarrow$  Executive

Client, Office  $\rightarrow$  Date

- a. Remove any existing partial dependencies and convert the logical schema to the Second Normal Form. Please remember that when performing schema decomposition you need to denote primary key for every new table as well as the foreign key that will allow us to reconstruct the original data.

=> CLIENT(Client, Date, Executive)

=> OFFICE(*Office*, Floor, Building, City)

[Note: Underline = PRIMARY KEY; Italic = FOREIGN KEY]

- b. Remove any existing transitive dependencies to create a set of logical schemas in Third Normal Form. Again, remember to denote primary keys and foreign keys (including which primary key those foreign keys point to).

City is a transitive dependency on Building so it can be removed.

=> CLIENT(Client, Date, Executive)

=> OFFICE(*Office*, Floor, Building)

[Note: Underline = PRIMARY KEY; Italic = FOREIGN KEY]

**Part 4**

Consider a table that stores information about students, student name, GPA, honors list and the credits that the student had completed so far.

(First, Last, GPA, Honor, Credits)

You are given the following functional dependencies

First, Last  $\rightarrow$  GPA, Honor, Credits

GPA  $\rightarrow$  Honor

- a. Is this schema in Second Normal Form? If not, please state which FDs violate 2NF and decompose the schema accordingly.

**Yes**

- b. Is this schema in Third Normal Form? If not, please state which FDs violate 3NF and decompose the schema accordingly.

Honor is a transitive dependency on GPA, so it can be removed from the STUDENTS table.

$\Rightarrow$  STUDENTS(First, Last, GPA, Credits)

[Note: Underline = PRIMARY KEY; Italic = FOREIGN KEY]

Be sure that your name and “Assignment 1” appear at the top of your submitted file.