CSC 455: Database Processing for Large-Scale Analytics
Assignment 2

Due 11:00pm, Friday, October 9th

Supplemental reading: SQL reference book *Oracle 11g SQL* by Price, **ISBN 9780071498500** (available in Books 24x7 DePaul online library as eBook). Sections 2.1 – 2.3, 2.6, 2.9 – 2.15 (section names are included in the screenshot→)

Python for Data Analysis, pp174 "Interacting with Databases"

You can read Chapter 6: Database Design Using Normalization in "Databases: A Beginner's Guide" by Andrew Oppel, **ISBN 0071608478** in the online library. Another good resource is in this book link.

Part 1

You are given a following schema in 1NF:
(License Number, Renewed, Status, Status Date, Driver Type, License Type, Original Issue Date, Name, Sex, Chauffer City, Chauffer State, Record Number) and the following functional dependencies:

Chauffeur City → Chauffeur State (both of these are a single column, not two columns)
Record Number → License Number, Renewed, Status, Status Date, Driver Type, License Type, Original Issue Date, Name, Sex, Chauffer City, Chauffer State

The table is based on a real data set original taken from City of Chicago data portal (located here: https://data.cityofchicago.org/Community-Economic-Development/Public-Chauffeurs/97wa-y6ff) However, the data has been cleaned and reduced to approximately one thousand rows. We will revisit a non-clean version of that data later.

Decompose the schema to make sure it is in Third Normal Form (3NF).

Write SQL DDL to create the 3NF tables you created. Remember to declare primary and foreign keys as necessary in your SQL code.

Part 2

Write a python script that is going to create your tables from Part 1 in SQLite and populate them with data automatically. The data file is posted in Assignment 2 dropbox folder on D2L as Public_Chauffeurs_Short.csv
Use sqlite3 database as shown in class but remember to make data type changes to your tables from Part 1 (i.e., NUMBER(5,0)→INTEGER, NUMBER(5,2)→REAL). SQLite is very forgiving regarding data types, but most databases are not.

I have some sample code that connects to a SQLite database, loads comma-separated student data and prints the contents of the loaded table. You can find it in the Assignment2 dropbox folder as loadStudentData.py and a Students.txt file that goes with it.
Naturally you would have to populate however many tables you have created in Part 1, not just 1 table.

For this assignment only, if you run into primary key conflict when loading data (i.e., "sqlite3.IntegrityError: column ID is not unique" error), you may use INSERT OR IGNORE instead of INSERT when loading data. This will cause INSERT to skip over duplicate inserts without causing an error.

Remember to load NULLs properly (i.e. not as string) and make sure you do not load the very first line that contains column names.

Part 3

You were hired to do some data analysis for a local zoo. Below is the data table, including the necessary constraints and all the insert statements to populate the database.

```
-- Drop all the tables to clean up
DROP TABLE Animal;

-- ACategory: Animal category 'common', 'rare', 'exotic'.  May be NULL
-- TimeToFeed: Time it takes to feed the animal (hours)
CREATE TABLE Animal
(
  AID      NUMBER(3, 0),
  AName      VARCHAR2(30) NOT NULL,
  ACategory VARCHAR2(18),

  TimeToFeed NUMBER(4,2),

  CONSTRAINT Animal_PK
    PRIMARY KEY(AID)
);
```

```
INSERT INTO Animal VALUES(1, 'Galapagos Penguin', 'exotic', 0.5);
INSERT INTO Animal VALUES(2, 'Emperor Penguin', 'rare', 0.75);
INSERT INTO Animal VALUES(3, 'Sri Lankan sloth bear', 'exotic', 2.5);
INSERT INTO Animal VALUES(4, 'Grizzly bear', NULL, 2.5);
INSERT INTO Animal VALUES(5, 'Giant Panda bear', 'exotic', 1.5);
INSERT INTO Animal VALUES(6, 'Florida black bear', 'rare', 1.75);
INSERT INTO Animal VALUES(7, 'Siberian tiger', 'rare', 3.75);
INSERT INTO Animal VALUES(8, 'Bengal tiger', 'common', 2.75);
INSERT INTO Animal VALUES(9, 'South China tiger', 'exotic', 2.25);
INSERT INTO Animal VALUES(10, 'Alpaca', 'common', 0.25);
INSERT INTO Animal VALUES(11, 'Llama', NULL, 3.5);
```

Since none of the managers in the zoo know SQL, it is up to you to write the queries to answer the following list of questions.

1. Find all the animals (their names) that take less than 1.5 hours to feed.
2. Find all the rare animals and sort the query output by feeding time (any direction)
3. Find the animal names and categories for animals related to a bear (hint: use the LIKE operator)
4. Return the listings for all animals whose rarity is not specified in the database
5. Find the rarity rating of all animals that require between 1 and 2.5 hours to be fed
6. Find the names of the animals that are related to the tiger and are not common
7. Find the minimum and maximum feeding time amongst all the animals in the zoo (single query)
8. Find the average feeding time for all the rare animals
9. Find listings for animals with ID less than 10 and also require more than 2 hours to feed.

Be sure that your name and “Assignment 2” appear at the top of your submitted file.