Alex Teboul

DSC 450: Database Processing for Large-Scale Analytics

Assignment 2

Due Sunday, April 28th

Supplemental reading: SQL reference book *Oracle 11g SQL* by Price, ISBN 9780071498500 (available in Books 24x7 DePaul online library as eBook). Sections:
Pp.31-34, "Performing Arithmetic"
pp.35-40, "Combining Column Output Using Concatentation", "Null Values", "Displaying Distinct Rows", "Comparing Values", "Using the LIKE Operator"
pp. 43- 46, "Using the Logical Operators", "Sorting Rows Using the ORDER BY Clause"

For a normalization reference, you can read Section 14.6 in "Information modeling and relational databases", **ISBN 9780123735683** in the online library (although in this class we only consider 1NF/2NF/3NF).

Part 1

You are given a following schema in 1NF:
(<u>First</u>, <u>Last</u>, Address, <u>Job</u>, Salary, Assistant) and the following set of functional dependencies:

First, Last → Address
Job → Salary, Assistant

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

- 3 Tables
- <u>primary key</u>, *foreign key*
- Partial FDs eliminated, no Transitive FDs
  - Employees(<u>First</u>, <u>Last</u>, Address)
  - Jobs(<u>Job</u>, Salary, Assistant)
  - IDs(<u>*FirstName*</u>, <u>*LastName*</u>, <u>*JobName*</u>)
- IDs(FirstName, LastName) reference Employees(First,Last)
- IDs(JobName) reference Jobs(Job)

Write SQL DDL and SQL INSERT statements to create the 3NF tables and manually load the data provided in the text file data_hw2.txt (posted together with this assignment). Make sure that NULL in the data file is loaded as database NULL and not as 'NULL' string.

```
CREATE TABLE Employees (
   First VARCHAR2(32) NOT NULL,
   Last VARCHAR2(32) NOT NULL,
   Address VARCHAR2(64),

   CONSTRAINT Employee_PK
      PRIMARY KEY(First, Last)
);

CREATE TABLE  Jobs (
        Job VARCHAR2(32) NOT NULL,
   Salary VARCHAR2(3),
        Assistant VARCHAR2(3),

        CONSTRAINT Jobs_PK
      PRIMARY KEY(Job)
);

CREATE TABLE IDs (
   FirstName VARCHAR2(32) NOT NULL,
   LastName VARCHAR2(32) NOT NULL,
   JobName VARCHAR2(32) NOT NULL,

   CONSTRAINT IDs_PK
      PRIMARY KEY(FirstName,LastName,JobName),
   CONSTRAINT IDs_FK1
```

```
        FOREIGN KEY(FirstName,LastName) REFERENCES
EMPLOYEES(First,Last),
    CONSTRAINT IDs_FK2
        FOREIGN KEY(JobName) REFERENCES Jobs(Job)
);


INSERT INTO Employees VALUES('John','Smith','111 N. Wabash Avenue');
INSERT INTO Employees VALUES('Jane','Doe','243 S. Wabash Avenue');
INSERT INTO Employees VALUES('Mike','Jackson', '1 Michigan Avenue');
INSERT INTO Employees VALUES('Mary','Who', '20 S. Michigan Avenue');

SELECT * FROM employees; --checks that it worked

INSERT INTO Jobs VALUES ('plumber', '40K', NULL);
INSERT INTO Jobs VALUES('bouncer', '35K', NULL);
INSERT INTO Jobs VALUES('waitress', '50K', 'Yes');
INSERT INTO Jobs VALUES('accountant', '42K', 'Yes');
INSERT INTO Jobs VALUES('bouncer', '35K', NULL);
INSERT INTO Jobs VALUES('accountant', '42K', 'Yes');
INSERT INTO Jobs VALUES('plumber', '40K', NULL);
INSERT INTO Jobs VALUES('accountant', '42K', 'Yes');
INSERT INTO Jobs VALUES('risk analyst', '80K', 'Yes');

SELECT * FROM Jobs; --checks that it worked

INSERT INTO IDs VALUES ('John', 'Smith','plumber');
INSERT INTO IDs VALUES ('John', 'Smith','bouncer');
INSERT INTO IDs VALUES ('Jane', 'Doe','waitress');
INSERT INTO IDs VALUES ('Jane', 'Doe','accountant');
INSERT INTO IDs VALUES ('Jane', 'Doe','bouncer');
INSERT INTO IDs VALUES ('Mike', 'Jackson','accountant');
INSERT INTO IDs VALUES ('Mike', 'Jackson','plumber');
INSERT INTO IDs VALUES ('Mary', 'Who','accountant');
INSERT INTO IDs VALUES ('Mary', 'Who','risk analyst');

SELECT * FROM IDs; --checks that it worked
```

Part 2

Write a python script that is going to create your tables from Part 1 and populate them with data automatically. Use sqlite3 database and make the necessary data type changes in your DDL from part1 (e.g., NUMBER(5,0) → INTEGER, NUMBER(5,2) → REAL).

Your python code must successfully load data into all tables (however many tables you have created in Part-1). Note that you are loading data into SQLite, not into Oracle (the process is the same, but connecting to Oracle DB using python is somewhat complicated, so we use SQLite instead). Your python code must read the input file data_hw2.txt directly and automatically load the data. Do not create INSERT SQL statements manually instead reading the file.

```
#Alex Teboul
#Assignment 2
#Problem 2

import os
import sqlite3

conn=sqlite3.connect('dsc450a2.db')
os.getcwd()
c = conn.cursor()

#SQL DDL to Strings
employees = '''CREATE TABLE Employees (
   First VARCHAR2(32),
   Last VARCHAR2(32),
   Address VARCHAR2(64),

   CONSTRAINT Employee_PK
      PRIMARY KEY(First, Last)
);'''
```

```
jobs = '''CREATE TABLE  Jobs (
            Job VARCHAR2(32),
   Salary VARCHAR2(3),
            Assistant VARCHAR2(3),

            CONSTRAINT Jobs_PK
      PRIMARY KEY(Job)
);'''

ids = '''CREATE TABLE IDs (
   FirstName VARCHAR2(32),
   LastName VARCHAR2(32),
   JobName VARCHAR2(32),

   CONSTRAINT IDs_PK
      PRIMARY KEY(FirstName,LastName,JobName),
   CONSTRAINT IDs_FK1
      FOREIGN KEY(FirstName,LastName) REFERENCES EMPLOYEES(First,Last),
   CONSTRAINT IDs_FK2
      FOREIGN KEY(JobName) REFERENCES Jobs(Job)
);'''

#c.execute(employees)
#c.execute(jobs)
#c.execute(ids)

#get data
file = open('data_hw2.txt','r')
lines = file.readlines()
file.close()

for line in lines:
   clean_line = line.strip().replace('NULL', 'None')
   values = clean_line.split(', ')
   c.execute("INSERT OR IGNORE INTO employees VALUES (?, ?, ?);", [values[0], values[1], values[2]]);
#into employees
   c.execute("INSERT OR IGNORE INTO jobs VALUES (?, ?, ?);", [values[3], values[4], values[5]]); #into
jobs
   c.execute("INSERT OR IGNORE INTO ids VALUES (?, ?, ?);", [values[0], values[1], values[3]]); #into ids
conn.commit()
conn.close()
```

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', 'common', 3.0);
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', 3.25);
INSERT INTO Animal VALUES(8, 'Bengal tiger', 'common', 2.75);
INSERT INTO Animal VALUES(9, 'South China tiger', 'exotic', 2.5);
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.**
    a. SELECT AName,TimeToFeed FROM Animal WHERE TimeToFeed<1.5;--used to check
    b. SELECT AName FROM Animal WHERE TimeToFeed<1.5;

AName
Galapagos Penguin
Emperor Penguin
Alpaca

2. **Find all the rare animals and sort the query output by feeding time (from small to large)**
    a. SELECT AName,TimeToFeed FROM Animal WHERE ACategory = 'rare' ORDER BY TimeToFeed;

AName	**TimeToFeed**
Emperor Penguin	0.75
Florida black bear	1.75
Siberian tiger	3.25

3. **Find the animal names and categories for animals related to a bear (hint: use the LIKE operator)**
    a. SELECT AName,Acategory FROM Animal WHERE AName LIKE '%bear';

AName	**ACategory**
Sri Lankan sloth bear	exotic
Grizzly bear	common
Giant Panda bear	exotic
Florida black bear	rare

4. **Return the listings for all animals whose rarity is missing in the database**
    a. SELECT * FROM Animal WHERE ACategory is NULL;

AID	**AName**	**ACategory**	**TimeToFeed**
11	Llama	(null)	3.5

5. **Find the rarity rating of all animals that require between 1 and 2.5 hours to be fed**
    a. SELECT ACategory FROM Animal WHERE TimeToFeed BETWEEN 1 and 2.5;

ACategory
exotic
exotic

rare
exotic

6. **Find the names of the animals that are related to the tiger and are not common**
    a. SELECT AName FROM Animal WHERE AName LIKE '%tiger%' AND ACategory!='common';

AName
Siberian tiger
South China tiger

7. **Find the names of the animals that are <u>not</u> related to the tiger**
    a. SELECT AName FROM Animal WHERE Aname NOT LIKE '%tiger%';

AName
Galapagos Penguin
Emperor Penguin
Sri Lankan sloth bear
Grizzly bear
Giant Panda bear
Florida black bear
Alpaca
Llama

8. **Find the minimum and maximum feeding time amongst all the animals in the zoo**
    a. SELECT min(TimeToFeed),Max(TimeToFeed) FROM Animal;

min(TimeToFeed)	**max(TimeToFeed)**
0.25	3.5

9. **Find the average feeding time for all of the rare animals**
    a. SELECT AVG(TimeToFeed) FROM Animal WHERE ACategory='rare';

avg(TimeToFeed)
1.9166666666666666666666666666666667

<u>Be sure that your name and "Assignment 2" appear at the top of your submitted file</u>.