COMS W4111-002, V02 (Spring 2022)
Introduction to Databases

Homework 2: Non-Programming

Due Sunday, February 27, 2022 at 11:59 PM

Introduction

Overview

This homework has 1 section:

1. A section for non-programming track.

Submission

You will **submit 2 files** for this assignment.

1. Submit a zip file titled `<your_uni>_hw2_nonprogramming.zip` to **HW2 Non-Programming - Zip** on Gradescope.

    - Replace `<your_uni>` with your uni. My submission would be `dff9_hw2_nonprogramming.zip`.
    - The zipped directory should include:
        - TODO: include files in the hw directory
        - `<your_uni>_hw2_nonprogramming.ipynb` (substitute with your uni as above)
        - Any image files you embed in your notebook.

2. Submit a PDF title `<your_uni>_hw2_nonprogramming.pdf` to **HW2 Non-Programming - PDF** on Gradescope.

    - This should be a PDF of your completed HW2 Non-Programming Python notebook.
        - **Tag pages for each problem**. Per course policy, any untagged submission will receive an automatic 0.
        - Double check your submission on Gradescope to ensure that the PDF conversion worked and that your pages are appropriately tagged.

Collaboration and Information

- Answering some of the questions may require independent research to find information. We encourage you to try troubleshooting problems independently before reaching out for help.
- You may use any information you get in TA or Prof. Ferguson's office hours, from lectures or from recitations. This includes slides related to the recommended textbook.
- You may use information that you find on the web.
- You are NOT allowed to collaborate with other students outside of office hours.

Non-Programming

Setup

- Modify the cells below to setup your environment.
- The change should just be setting the DB user ID and password, replacing my user ID and password with yours for MySQL.

In [1]:
```
database_user_id = "root"
database_pwd = "Cherry_0127"
```

In [2]:
```
database_url = "mysql+pymysql://" + \
    database_user_id + ":" + database_pwd + "@localhost"
database_url
```

Out[2]:
```
'mysql+pymysql://root:Cherry_0127@localhost'
```

In [3]:
```
%reload_ext sql
```

In [4]:
```
%sql $database_url
```

Out[4]:
```
'Connected: root@None'
```

In [5]:
```
from sqlalchemy import create_engine
```

In [6]:
```
sqla_engine = create_engine(database_url)
```

In [7]:
```
#
# We are going to create a schema and some tables for the HW.
#
%sql create schema if not exists S22_W4111_HW2_B
%sql select 1;
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[7]:

1
1

Install Datasets

Classic Models

- We will use the [Classic Models Tutorial (https://www.mysqltutorial.org/mysql-sample-database.aspx)](https://www.mysqltutorial.org/mysql-sample-database.aspx) database for HW 2 Non-Programming, other homework assignments, and exams.
- Lecture 5 briefly explained why this data model is interesting for education purposes. The problems on homework assignments and exams will further explore why it's interesting.
- The zip file for HW 2 Non-Programming contains an SQL script for creating a database `classicmodels` and loading the data. The script is `classicmodels.sql`.
- Use DataGrip to run the script. You performed this task for HW 0 with different SQL scripts. The basic approach is:
  - Right click on `@localhost`
  - Choose `Run SQL Script`.
  - Navigate to and select `classicmodels.sql`.
- The following cells test for correct installation.
- These cells are also examples of DDL statements and querying the "catalog."

In [11]:
```
%sql show tables from classicmodels
```

```
 * mysql+pymysql://root:***@localhost
8 rows affected.
```

Out[11]:

Tables_in_classicmodels
customers
employees
offices
orderdetails
orders
payments
productlines
products

In [110]:
```
%%sql

select
    table_schema, table_name, column_name, IS_NULLABLE, DATA_TYPE from info
where
    table_schema='classicmodels'
order by
    table_schema, table_name, ORDINAL_POSITION
limit 10;
```

```
 * mysql+pymysql://root:***@localhost
10 rows affected.
```

Out[110]:

TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	IS_NULLABLE	DATA_TYPE
classicmodels	customers	customerNumber	NO	int
classicmodels	customers	customerName	NO	varchar
classicmodels	customers	contactLastName	NO	varchar
classicmodels	customers	contactFirstName	NO	varchar
classicmodels	customers	phone	NO	varchar
classicmodels	customers	addressLine1	NO	varchar
classicmodels	customers	addressLine2	YES	varchar
classicmodels	customers	city	NO	varchar
classicmodels	customers	state	YES	varchar
classicmodels	customers	postalCode	YES	varchar

In [109]:
```
%%sql
use classicmodels;
with
    customer_orders_details as
        (
            select customerNumber, orderNumber, status, orderDate, shippedD
                   productCode, quantityOrdered, priceEach
            from orders natural join orderdetails
        ),
     customer_orders_totals as
         (
             select customerNumber, orderNumber,
                    concat(
                            '$',
                            format(sum(priceEach * quantityOrdered), 2)
                        ) as order_value
                from customer_orders_details
                group by customerNumber, orderNumber
         )
select * from customer_orders_totals
limit 10;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
10 rows affected.
```

Out[109]:

customerNumber	orderNumber	order_value
103	10123	$14,571.44
103	10298	$6,066.78
103	10345	$1,676.14
112	10124	$32,641.98
112	10278	$33,347.88
112	10346	$14,191.12
114	10120	$45,864.03
114	10125	$7,565.08
114	10223	$44,894.74
114	10342	$40,265.60

World, Country, State, City

- Having definitive information about countries, cities, etc. is useful for data engineer and data insight.
- We will use information from Darshan Gada's GitHub project (https://github.com/dr5hn). For convenience, I have copied SQL scripts into the homework directory.
- Use DataGrip to create a schema `world city state` .

- Select the newly created schema and right click to choose `Run SQL Script` to run the scripts:
  - `world_city_state_countries.sql`
  - `world_city_state_states.sql`
  - `world_city_state_cities.sql`

Copy Information

- We want to preserve the original data. So, we will copy the data and **structure** into the HW 2 B database.
- Set the current database to `S22_W4111_HW2_B` .
- Create tables in the database for every table in `classicmodels` and `world_city_state` .
- Load the data into the new tables from the original tables.
- The tables in `S22_W4111_HW2_B` **MUST** have the same column names, types, constraints, etc.
- You **MUST** perform this task by executing SQL statements in cells below.
- This task may seem overly tedious and complex. But, if you think about it, you will realize that writing many of the statements from scratch is not necessary.

In [20]:
```
%%sql

use S22_W4111_HW2_B;
select 1;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1 rows affected.
(pymysql.err.ProgrammingError) (1064, 'You have an error in your SQL synt
ax; check the manual that corresponds to your MySQL server version for th
e right syntax to use near \'<"classicmodels">\' at line 1')
[SQL: SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA
= <"classicmodels">;]
(Background on this error at: https://sqlalche.me/e/14/f405) (https://sql
alche.me/e/14/f405))
```

In [25]:
```
%%sql

SELECT
 table_name
FROM INFORMATION_SCHEMA.TABLES
WHERE
 TABLE_SCHEMA = 'classicmodels';
```

```
 * mysql+pymysql://root:***@localhost
8 rows affected.
```

Out[25]:

TABLE_NAME
customers
employees
offices
orderdetails
orders
payments
productlines
products

In [26]:
```
%%sql

SELECT
 table_name
FROM INFORMATION_SCHEMA.TABLES
WHERE
 TABLE_SCHEMA = 'world_city_state';
```

```
 * mysql+pymysql://root:***@localhost
3 rows affected.
```

Out[26]:

TABLE_NAME
cities
countries
states

In [30]:

```
%%sql

CREATE
 TABLE S22_W4111_HW2_B.cities LIKE world_city_state.cities;
INSERT INTO S22_W4111_HW2_B.cities
 SELECT * FROM world_city_state.cities;

CREATE
 TABLE S22_W4111_HW2_B.countries LIKE world_city_state.countries;
INSERT INTO S22_W4111_HW2_B.countries
 SELECT * FROM world_city_state.countries;

CREATE
 TABLE S22_W4111_HW2_B.states LIKE world_city_state.states;
INSERT INTO S22_W4111_HW2_B.states
 SELECT * FROM world_city_state.states;

CREATE
 TABLE S22_W4111_HW2_B.customers LIKE classicmodels.customers;
INSERT INTO S22_W4111_HW2_B.customers
 SELECT * FROM classicmodels.customers;

CREATE
 TABLE S22_W4111_HW2_B.employees LIKE classicmodels.employees;
INSERT INTO S22_W4111_HW2_B.employees
 SELECT * FROM classicmodels.employees;

CREATE
 TABLE S22_W4111_HW2_B.offices LIKE classicmodels.offices;
INSERT INTO S22_W4111_HW2_B.offices
 SELECT * FROM classicmodels.offices;

CREATE
 TABLE S22_W4111_HW2_B.orderdetails LIKE classicmodels.orderdetails;
INSERT INTO S22_W4111_HW2_B.orderdetails
 SELECT * FROM classicmodels.orderdetails;

CREATE
 TABLE S22_W4111_HW2_B.orders LIKE classicmodels.orders;
INSERT INTO S22_W4111_HW2_B.orders
 SELECT * FROM classicmodels.orders;

CREATE
 TABLE S22_W4111_HW2_B.payments LIKE classicmodels.payments;
INSERT INTO S22_W4111_HW2_B.payments
 SELECT * FROM classicmodels.payments;

CREATE
 TABLE S22_W4111_HW2_B.productlines LIKE classicmodels.productlines;
INSERT INTO S22_W4111_HW2_B.productlines
 SELECT * FROM classicmodels.productlines;

CREATE
 TABLE S22_W4111_HW2_B.products LIKE classicmodels.products;
INSERT INTO S22_W4111_HW2_B.products
 SELECT * FROM classicmodels.products;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
148048 rows affected.
0 rows affected.
250 rows affected.
0 rows affected.
4963 rows affected.
0 rows affected.
122 rows affected.
0 rows affected.
23 rows affected.
0 rows affected.
7 rows affected.
0 rows affected.
2996 rows affected.
0 rows affected.
326 rows affected.
0 rows affected.
273 rows affected.
0 rows affected.
7 rows affected.
0 rows affected.
110 rows affected.
```

Out[30]: []

Data Transformation

- The query below shows some information from `classicmodels.customers` .

In [108]:
```
%sql select * from classicmodels.customers limit 10;
```

```
 * mysql+pymysql://root:***@localhost
10 rows affected.
```

Out[108]:

customerNumber	customerName	contactLastName	contactFirstName	phone	addressLine1	a
103	Atelier graphique	Schmitt	Carine	40.32.2555	54, rue Royale	
112	Signal Gift Stores	King	Jean	7025551838	8489 Strong St.	
114	Australian Collectors, Co.	Ferguson	Peter	03 9520 4555	636 St Kilda Road	
119	La Rochelle Gifts	Labrune	Janine	40.67.8555	67, rue des Cinquante Otages	
121	Baane Mini Imports	Bergulfsen	Jonas	07-98 9555	Erling Skakkes gate 78	
124	Mini Gifts Distributors Ltd.	Nelson	Susan	4155551450	5677 Strong St.	
125	Havel & Zbyszek Co	Piestrzeniewicz	Zbyszek	(26) 642-7555	ul. Filtrowa 68	
128	Blauer See Auto, Co.	Keitel	Roland	+49 69 66 90 2555	Lyonerstr. 34	
129	Mini Wheels Co.	Murphy	Julie	6505555787	5557 North Pendale Street	
131	Land of Toys Inc.	Lee	Kwai	2125557818	897 Long Airport Avenue	

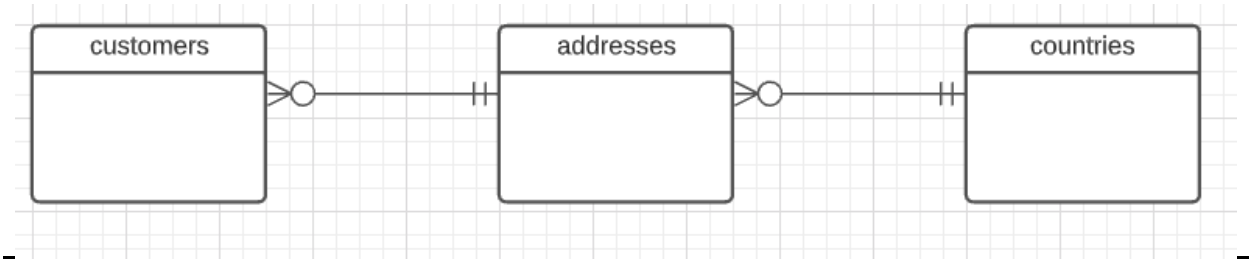
- There are several problems with this table definition, but we will focus on two.

1. Directly storing values like a country's or city's name is error prone. For example, different users and applications could enter various values:
   - Country: "United States," "USA," "US," etc.
   - City: "NYC," "New York," etc.

2. Having address information in rows with company information can cause errors and ambiguity over time, e.g.
   - There are cases where multiple companies have the same address, or a company has multiple addresses.
   - Just because a company "goes away" does not mean the address "went away."

- To fix these problems, you must transform the schema and data. This task will also require some data cleanup.

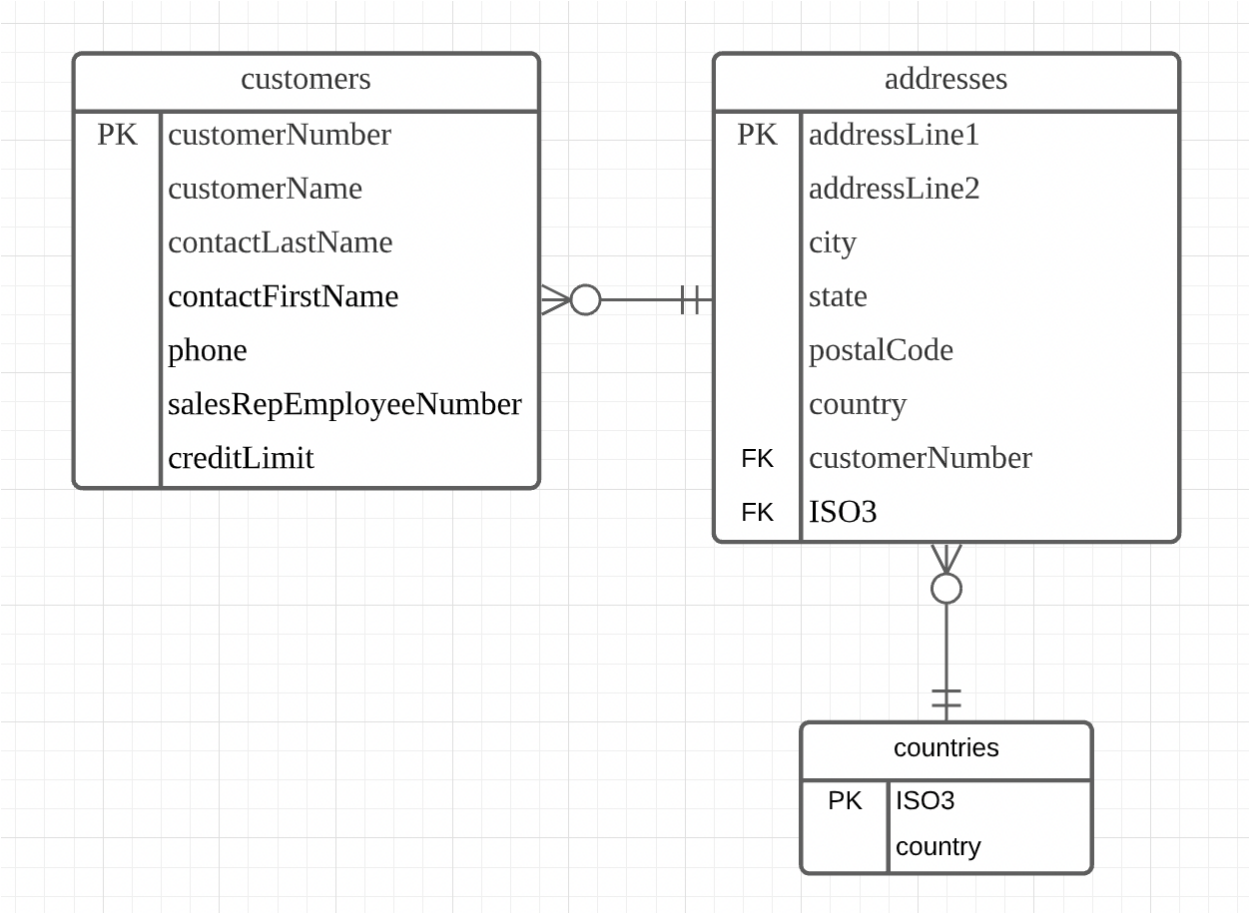
- The conceptual model you should implement is:



Customers-Address Conceptual Model

- You have to determine how to connect/link the tables. While you may include columns in one table that contain values in another table, do not worry about formally setting foreign key constraints. The important think is that you understand how they're linked.
- A good design would also handle ambiguity over city, state, etc. names. You do not need to worry about anything other than removing addresses from customers and handling countries.
- In the cells below, enter your SQL statements for creating and modifying tables, and modify data.

The structure of the model I want to build is:



In [34]:
```
%%sql
# Rename the original customers table
rename table customers to customers_origin;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

Out[34]: []

In [36]:
```
%%sql
# Create new customers table
create table customers
(
    customerNumber         int           not null,
    customerName           VARCHAR(512) null,
    contactLastName        VARCHAR(256) null,
    contactFirstName       VARCHAR(256) null,
    phone                  VARCHAR(256) null,
    salesRepEmployeeNumber int           null,
    creditLimit            numeric       null,
    primary key (customerNumber)
);
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

Out[36]: []

In [44]:
```
%%sql
# Copy info into customers
INSERT INTO
    S22_W4111_HW2_B.customers(customerNumber, customerName, contactLastName
SELECT
    customerNumber,
    customerName,
    contactLastName,
    contactFirstName,
    phone,
    salesRepEmployeeNumber,
    creditLimit
FROM customers_origin
```

```
 * mysql+pymysql://root:***@localhost
122 rows affected.
```

Out[44]: []

In [75]:
```
%%sql
# Find those country_names in customers_origin which cannot be found in cou
Select distinct country, c.iso3 from customers_origin co
left join countries c
on c.name = co.country
where c.iso3 is null
```

```
 * mysql+pymysql://root:***@localhost
3 rows affected.
```

Out[75]:

country	iso3
USA	None
UK	None
Hong Kong	None

In [88]:
```
%%sql
# Add a new column ISO3 into customers_origin
alter table customers_origin
    add ISO3 varchar(3) null;
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

Out[88]: []

In [89]:
```
%%sql
UPDATE customers_origin co
LEFT JOIN countries c
ON c.name = co.country
SET co.ISO3 = c.iso3
```

```
 * mysql+pymysql://root:***@localhost
122 rows affected.
```

Out[89]: []

In [90]:
```
%%sql
UPDATE customers_origin co
SET co.ISO3 =
CASE
    WHEN co.country = "USA" THEN "USA"
    WHEN co.country = "UK" THEN "GBR"
    WHEN co.country = "Hong Kong" THEN "HKG"
END

WHERE co.country IN ('USA', 'UK', 'Hong Kong')
```

```
 * mysql+pymysql://root:***@localhost
42 rows affected.
```

Out[90]: []

In [93]:
```
%%sql
# Create countries_new
create table countries_new
(
    ISO3     varchar(3)   not null,
    country  varchar(100) null,
    primary key (ISO3)
);
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

Out[93]: []

In [96]:
```
%%sql
# Copy info from customers_origin
INSERT INTO
    S22_W4111_HW2_B.countries_new(ISO3, country)
SELECT
    distinct ISO3,
    country
from customers_origin
```

```
 * mysql+pymysql://root:***@localhost
27 rows affected.
```

Out[96]: []

In [100]:
```
%%sql
# Create new addresses table
create table addresses
(
    customerNumber int          not null,
    addressLine1   varchar(512) not null,
    addressLine2   varchar(256) null,
    city           varchar(64)  null,
    state          varchar(32)  null,
    postalCode     varchar(32)  null,
    country        varchar(100)  null,
    ISO3           varchar(8)   null,
    primary key (addressLine1),
    foreign key (customerNumber) references customers(customerNumber),
    foreign key (ISO3) references countries_new(ISO3)
);
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```

Out[100]: []

In [101]:
```
%%sql
# Copy info into addresses
INSERT INTO
    S22_W4111_HW2_B.addresses(customerNumber, addressLine1, addressLine2, c
SELECT
    customerNumber,
    addressLine1,
    addressLine2,
    city,
    state,
    postalCode,
    country,
    ISO3
FROM customers_origin
```

```
 * mysql+pymysql://root:***@localhost
122 rows affected.
```

Out[101]:
```
[]
```

- Put SQL statements in the cell below to return information about customers, including address.

In [107]:
```
%%sql
SELECT * FROM customers c
LEFT JOIN addresses a
ON a.customerNumber = c.customerNumber
LIMIT 10;
```

```
 * mysql+pymysql://root:***@localhost
10 rows affected.
```

Out[107]:

customerNumber	customerName	contactLastName	contactFirstName	phone	salesRepEmploy
103	Atelier graphique	Schmitt	Carine	40.32.2555	
112	Signal Gift Stores	King	Jean	7025551838	
114	Australian Collectors, Co.	Ferguson	Peter	03 9520 4555	
119	La Rochelle Gifts	Labrune	Janine	40.67.8555	
121	Baane Mini Imports	Bergulfsen	Jonas	07-98 9555	
124	Mini Gifts Distributors Ltd.	Nelson	Susan	4155551450	
125	Havel & Zbyszek Co	Piestrzeniewicz	Zbyszek	(26) 642-7555	
128	Blauer See Auto, Co.	Keitel	Roland	+49 69 66 90 2555	
129	Mini Wheels Co.	Murphy	Julie	6505555787	
131	Land of Toys Inc.	Lee	Kwai	2125557818	

In [ ]: