Mini-Project #2

Objectives of Lab 2

1. Parse a large 'messy' file
2. Create a normalized database from a large 'messy' file
3. Optimize code to load 600,000+ rows into a database quickly
4. Practice SQL queries

Description of data

Open `lab2_data.csv` file and understand its contents. The file consists of 11 columns separated by tab and each row is a customer with all of the products they have ordered. The columns are:

1. Name -- The `Name` column is made up of the customer `FirstName` and `LastName` separated by a space
2. Address -- The `Address` column has the customer address
3. City -- The `City` column has the customer city
4. Country -- The `Country` column has the customer country
5. Region -- The `Region` column has the customer region
6. ProductName*
7. ProductCategory*
8. ProductCategoryDescription*
9. ProductUnitPrice*
10. QuantityOrderded*
11. OrderDate*

All the * columns are linked, meaning if you split the columns on a semicolon and zip them up, you will get one product name, product category, product category description, product unit price, quantity ordered, and ordered date. Remember, each row/line in the data represents a customer and all their product orders. The product orders are separated by a semicolon.

In this lab you will parse the data file and create a normalized database with six tables. To make this lab manageable, I will lay down the steps you need to create the following tables:

```
Region
  [RegionID] Integer not null primary key
  [Region] Text not null
Country
  [CountryID] integer not null Primary key
  [Country] Text not null
  [RegionID] integer not null foreign key to Region table
Customer
  [CustomerID] integer not null Primary Key
  [FirstName] Text not null
  [LastName] Text not null
  [Address] Text not null
  [City] Text not null
  [CountryID] integer not null foreign key to Country table
ProductCateogry
  [ProductCategoryID] integer not null Primary Key
  [ProductCategory] Text not null
  [ProductCategoryDescription] Text not null
Product
  [ProductID] integer not null Primary key
  [ProductName] Text not null
  [ProductUnitPrice] Real not null
  [ProductCategoryID] integer not null foreign key to ProductCateogry ta
OrderDetail
  [OrderID] integer not null Primary Key
  [CustomerID] integer not null foreign key to Customer table
  [ProductID] integer not null foreign key to Product table
  [OrderDate] integer not null
  [QuantityOrdered] integer not null
```

Hints

Note: The `create_table` function has been updated. Please study it and try to understand how it is different. You can now use it to drop a table before inserting into it. This is useful because you do not have to start from scratch. You can drop an individual table before recreating it if it exists.

Also checkout out this link to understand how `executemany` works: https://www.kite.com/python/docs/sqlite3.Cursor.executemany and incorporate it into your insert_table functions. If you use `execute`, your insertions will be very slow. If you have only one value to insert, the values tuple will look like

```
values = (('Graduate', ), ('Undergraduate',))
```

Always, always use the `with` context to insert.

Note: If you do not use `executemany`, I will take points off

DO NOT USE PANDAS to manipulate the data; I will deduct points for using PANDAS;

DO NOT USE CSV module to read the data; I will deduct points for using the CSV module;

Step 1 - Create the Region Table

Tasks

1. Loop over the data and find the distinct regions
2. Create the region table
3. Populate the region table with the distinct regions. Make sure to sort the regions before inserting.

Step 2 - Create dictionary to map Region to RegionID

The purpose of this step is to create a dictionary to look up the primary key lookup for a given region.

Tasks

1. Write an SQL query to fetch all the rows from the Region table
2. Transform the row into a dictionary whose key is Region and value is RegionID

Step 3 - Create the Country Table

Note: You will be using `region_to_regionid_dict` in this step when inserting into the country table. You can access `region_to_regionid_dict` by calling the function in step2 inside step3.

Tasks

1. Loop over the data and find the distinct countries and region combo
2. Create the country table

3. Populate the country table with the distinct countries and regio combo. Make sure to sort the countries before inserting.

Step 4 - Create dictionary to map Country to CountryID

The purpose of this step is to create a dictionary to look up the primary key lookup for a given country.

Tasks

1. Write an SQL query to fetch all the rows from the Country table
2. Transform the row into a dictionary whose key is Country and value is CountryID

Step 5 - Create the customer table

Note: You will be `country_to_countryid_dict` in this step when inserting into the customer table. You can access `country_to_countryid_dict` by calling the function in step4 inside step5.

Tasks

1. Loop over the data file and grab the name, address, city, country and region
2. Create the customer table
3. Populate the customer table

NOTE!

- One of name when split on space results in three elements. In this case, the first name is the first element and the other two elements in the last name.
- Insert the names sorted by first and lastname as one string. Hint: user lambda to combine first and last name and sort.

Step 6 - Create dictionary to map Name (FirstName LastName) to CustomerID

The purpose of this step is to create a dictionary to look up the primary key lookup for a given name (FirstName LastName)

Tasks

1. Write an SQL query to fetch all the rows from the Customer table
2. Transform the row into a dictionary whose key is the name of the customer (FirstName LastName) and value is CustomerID

Step 7 - Create Product Category Table

Tasks

1. Loop over the data and find the distinct product categories
2. Create the product category table
3. Populate the product category table with the distinct product categories. Make sure to sort the categories before inserting.

Step 8 - Create dictionary to map productcategory to productcategoryid

The purpose of this step is to create a dictionary to look up the primary key lookup for a given product category

Tasks

1. Write an SQL query to fetch all the rows from the product category table
2. Transform the row into a dictionary whose key is the product category and value is product category id

Step 9 - Create Product Table

Note: You will be using `productcategory_to_productcategoryid_dict` in this step when inserting into the product table.

Tasks

1. Loop over the data and find the distinct products
2. Create the product table
3. Populate the product table with the distinct products. Make sure to sort the products (by name) before inserting.

Step 10 - Create dictionary to map product to productid

The purpose of this step is to create a dictionary to look up the primary key lookup for a given product

Tasks

1. Write an SQL query to fetch all the rows from the product table

2. Transform the row into a dictionary whose key is the product and value is product id

Step 11 - Create OrderDetail Table

Note: You will be using `product_to_productid_dict` and `customer_to_customerid_dict` in this step when inserting into the order detail table.

Tasks

1. Loop over the data and figure out the orders
2. Create the orderdetails table
3. Populate the orderdetails table with the orders

Hint

- convert orderdate to format '%Y-%m-%d'

```
import datetime.datetime
datetime.datetime.strptime(input_date, '%Y%m%d').strftime('%Y-%m-%d')
```

- convert quantity ordered to int!