Databázy SQL: Projekt

Vypracoval: Martin Častvan

Úvod

V tomto projekte sa budeme zaoberať exportovaním dát z excelovskej tabuľky do SQLite3 databázy. Pritom definujeme nasledovné kroky (**ELT job**)s dosiahnutiu hore spomenutých cieľov:

1. **Extract** - Extraktovanie dát z excelovského súboru
2. **Load** - Načítanie do Python Pandas Data frame a následne do SQLite databázy
3. **Transform** - Transformácia a čistenie dát pomocou SQL príkazov

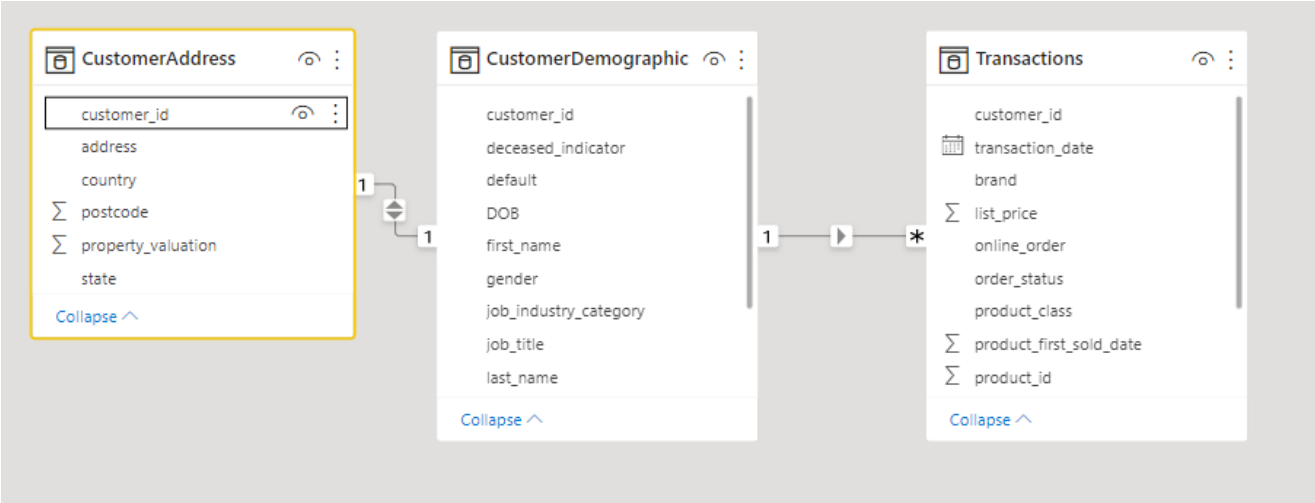
Dataset

Dáta pochádzajú z fiktívnej spoločnosti, ktorá sa zaoberá predávaním bicyklov. V excelovskom súbore sú tri relácie ktoré budeme chcieť uložiť do našej databázy.

1. **Transactions** - Relácia transakcii
2. **Customer Demographic**
3. **Customer Address** -

Všetky tieto relácie sú spojené tým istým poľom **customer_id**.

Data Model



Excel → Pandas Dataframe

Každú tabuľku nahráme do osobitného Pandas Dataframu.

```python
transactions = pd.read_excel("Transactions_dataset.xlsx", sheet_name=0)
demographic = pd.read_excel("Transactions_dataset.xlsx", sheet_name=2)
```

```
adresses = pd.read_excel("Transactions_dataset.xlsx", sheet_name=3)
```

SQLite3 Connection

V tomto projekte budeme pracovať s databázou, ktorá operuje iba v operačnej pamäti nášho počítača. Nie je to databáza, ktorá beží na nejakom vzdialenom servery. Dajú sa aj takéto riešenia použiť, ale je to nad rámec tohto projektu.

V Pythone urobíme urobíme jednoduchí connection na "marketing.db" súbor.

PYTHON

```
con = sqlite3.connect("marketing.db")
cursor = con.cursor()
```

Vytvorenie tabuliek v SQL

PYTHON

```
create_demographic = '''
    CREATE TABLE Demographic (
        customer_id INT,
        first_name TEXT,
        last_name TEXT,
        gender TEXT,
        past_3_years_bike_related_purchases INT,
        DOB TIMESTAMP,
        job_title TEXT,
        job_industry_category TEXT,
        wealth_segment TEXT,
        deceased_indicator INT,
        owns_car INT,
        tenure INT
    );
'''

cursor.execute(create_demographic)
```

PYTHON

```
create_adresses = '''
        CREATE TABLE "Adresses" (
        "index" INTEGER,
        "customer_id" INTEGER,
        "address" TEXT,
        "postcode" INTEGER,  "state" TEXT,
        "country" TEXT,
        "property_valuation" INTEGER
```

```
        )
'''

cursor.execute(create_adresses)
```

PYTHON

```
create_transactions = '''
    CREATE TABLE "Transactions" (
    "index" INTEGER,
    "transaction_id" INTEGER,
    "product_id" INTEGER,
    "customer_id" INTEGER,
    "transaction_date" TIMESTAMP,
    "online_order" REAL,
    "order_status" TEXT,
    "brand" TEXT,
    "product_line" TEXT,
    "product_class" TEXT,
    "product_size" TEXT,
    "list_price" REAL,
    "standard_cost" REAL,
    "product_first_sold_date" REAL
    )
'''

cursor.execute(create_transactions)
```

Vkladanie do databázy

Knižnica pandas umožňuje vložiť svoj dataframe do SQLite tabuľky. Robí sa to pomocou príkazu `DataFrame.to_sql("Meno_tabuľky_v_databaze", connection)`. Konkrétny príklad je demonštrovaný nasledovne:

PYTHON

```
transactions.to_sql("Transactions", con, if_exists="replace")

# stplec "default" je blbo definovny a neznamena nic. Preto ho treba
odstranit
demographic = demographic.drop(columns=["default"])
demographic.to_sql("Demographic", con, if_exists="replace")

adresses.to_sql("Adresses", con, if_exists="replace")
```

Nasledovným príkazom môžeme zistiť, koľko máme tabuliek v našej databázy a ktoré sa nahrali správne.

```python
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
cursor.fetchall()
```

```
# SHOW TABLES
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
cursor.fetchall()

[('Transactions',), ('Demographic',), ('Adresses',)]
```

Všetky tri tabuľky sa správne nahrali.

Pomocou `cursor.execute()` vieme štandardne dotazovať databázu. Problém je ale, že objekt ktorý nám táto metóda vrati je štandardný Python list. Tento objekt nepodporuje dátové operácie natívne.

```
cursor.execute("SELECT * FROM Adresses LIMIT 5;")
cursor.fetchall()

[(0, 1, '060 Morning Avenue', 2016, 'New South Wales', 'Australia', 10),
 (1, 2, '6 Meadow Vale Court', 2153, 'New South Wales', 'Australia', 10),
 (2, 4, '0 Holy Cross Court', 4211, 'QLD', 'Australia', 9),
 (3, 5, '17979 Del Mar Point', 2448, 'New South Wales', 'Australia', 4),
 (4, 6, '9 Oakridge Court', 3216, 'VIC', 'Australia', 9)]
```

Preto pre ďalšie spracovanie je odporúčané používanie príkazu `pandas.read_sql()` .

```
# We can use SQL queries and immediately output them into pandas dataframes for better viewing
cursor.execute("SELECT * FROM Demographic;")
cursor.fetchall()
selected_data = pd.read_sql("SELECT * FROM Demographic WHERE customer_id BETWEEN 100 AND 230",
                            con)
selected_data
```

	index	customer_id	first_name	last_name	gender	past_3_years_bike_related_purchases	DOB	job_
0	99	100	Tripp	Steed	Male	80	1968-05-11 00:00:00	Ger Man
1	100	101	Goran	Kwietek	Male	83	1975-12-29 00:00:00	N Practici
2	101	102	Langsdon	Tranfield	Male	51	1956-11-27 00:00:00	Qu Coı Spec
3	102	103	Ethyl	Runham	Female	49	1997-04-21 00:00:00	Pro Engi

Čistenie dát v SQLite

Keď sa pozrieme na pohlavie, vidíme že je reprezentované rôznymi hodnotami.

```
sql_query = '''
    SELECT DISTINCT(gender) FROM Demographic;
'''
cursor.execute(sql_query)
cursor.fetchall()
```

```
[('F',), ('Male',), ('Female',), ('U',), ('Femal',), ('M',)]
```

Takúto nezrovnalosť vieme odstrániť nasledovnými príkazmi:

PYTHON

```
update_query = '''
    UPDATE Demographic
    SET gender = 'M'
    WHERE gender = 'Male'
'''

cursor.execute(update_query)
con.commit()
```

PYTHON

```
update_query = '''
    UPDATE Demographic
    SET gender = 'F'
    WHERE gender IN ('Female', 'Femal')
'''
cursor.execute(update_query)
con.commit()
```

Po spustený nasledovného príkazu sa už dátové nezrovnalosti nevidia.

```
sql_query = '''
    SELECT DISTINCT(gender) FROM Demographic;
'''
cursor.execute(sql_query)
cursor.fetchall()
```

```
[('F',), ('M',), ('U',)]
```

Taktiež opravíme nasledovné dátové problémy.

```
cursor.execute('''
    SELECT DISTINCT(state) from Adresses;
''')
cursor.fetchall()
```

```
[('New South Wales',), ('QLD',), ('VIC',), ('NSW',), ('Victoria',)]
```

Niektoré štáty majú trojpísmenovú skratku a niektoré nie. Opravíme to nasledovne.

PYTHON

```
cursor.execute('''
    UPDATE Adresses
    SET state = 'New South Wales'
    WHERE state = 'NSW'
''')

cursor.execute('''
    UPDATE Adresses
    SET state = 'Victoria'
    WHERE state = 'VIC'
''')

cursor.execute('''
    UPDATE Adresses
    SET state = 'Queensland'
    WHERE state = "QLD"
''')

con.commit()
```

Vykreslenie grafov

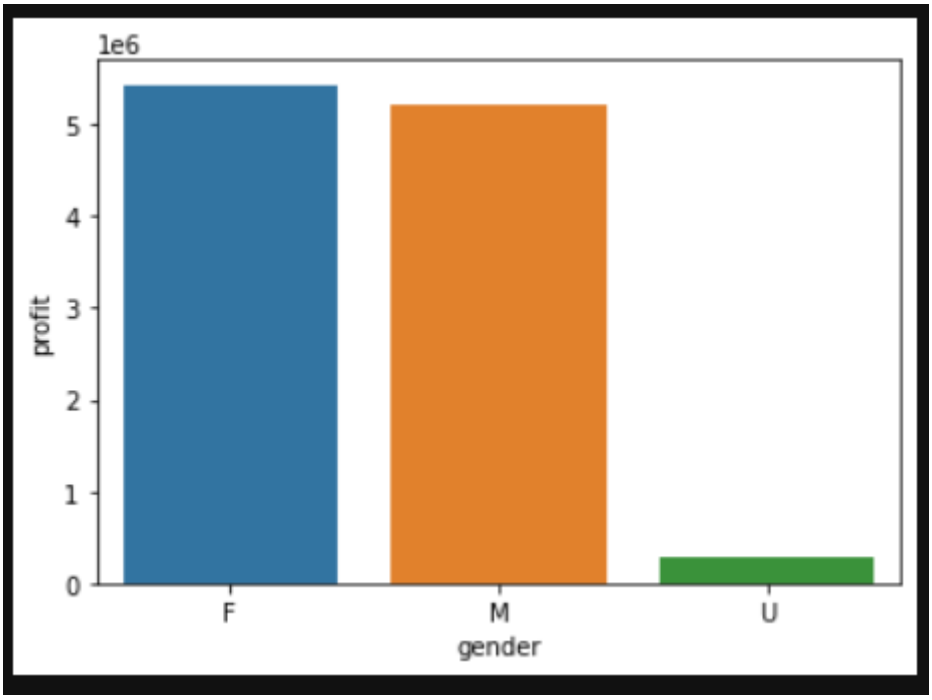
Profit v závislosti od pohlavia

Štandardne profit neexistuje v žiadnej relácii, preto si ho budeme musieť spočítať v SQL príkaze. Taktiež budeme musieť spojiť dve relácie do jednej (**Demographic** a **Transactions**).

PYTHON

```
joindata = pd.read_sql('''
    SELECT gender, SUM(list_price - standard_cost) as profit
    FROM Transactions t
    LEFT JOIN Demographic d ON t.customer_id = d.customer_id
    GROUP BY gender
''', con)

sns.barplot(x="gender", y="profit", data = joindata)
```



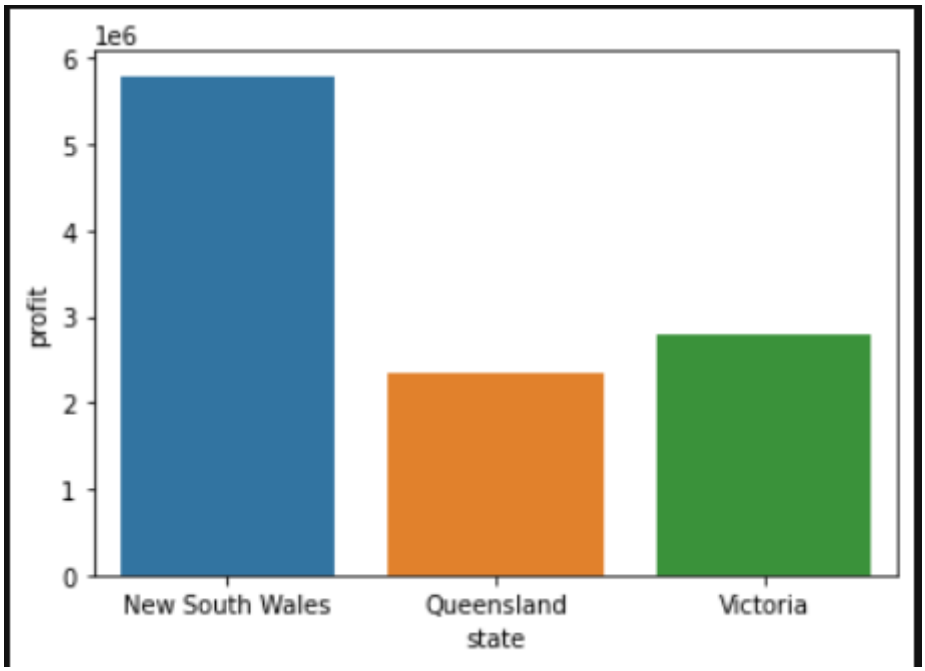
Profit v závislosti od štátu

Podobne ako v predchádzajúcom príklade, iba treba spojiť dve iné relácie (**Adresses** a **Transactions**).

PYTHON

```
joindata = pd.read_sql('''
    SELECT state, SUM(list_price - standard_cost) AS profit
    FROM Transactions
    LEFT JOIN Adresses ON Transactions.customer_id = Adresses.customer_id
    GROUP BY state
''', con)

sns.barplot(x='state', y='profit', data=joindata)
```



Záver

V tomto projekte sme si ukázali ako z excelovského súboru nahrať dáta do SQLite databázy, očistiť a následne ich vizualizovať pomocou Pythonu. Taktiež sme si ukázali niektoré praktiky pracovania s dátami.