

Lesson 2 Exercise 3 Creating Fact and Dimension Tables with Star Schema

January 21, 2023

1 Lesson 2 Exercise 3: Creating Fact and Dimension Tables with Star Schema

1.0.1 Walk through the basics of modeling data using Fact and Dimension tables. You will create both Fact and Dimension tables and show how this is a basic element of the Star Schema.

Where you see ##### you will need to fill in code.

1.0.2 This exercise will be more challenging than the last. Use the information provided to create the tables and write the insert statements.

1.0.3 Import the library

Note: An error might popup after this command has executed. If it does read it careful before ignoring.

```
In [1]: import psycopg2
```

1.0.4 Create a connection to the database

```
In [2]: try:
        conn = psycopg2.connect("host=127.0.0.1 dbname=studentdb user=student password=student")
    except psycopg2.Error as e:
        print("Error: Could not make connection to the Postgres database")
        print(e)
```

1.0.5 Next use that connect to get a cursor that we will use to execute queries.

```
In [3]: try:
        cur = conn.cursor()
    except psycopg2.Error as e:
        print("Error: Could not get cursor to the Database")
        print(e)
```

For this demo we will use automatic commit so that each action is committed without having to call `conn.commit()` after each command. The ability to rollback and commit transactions is a feature of Relational Databases.

```
In [4]: conn.set_session(autocommit=True)
```

1.0.6 Imagine you work at an online Music Store. There will be many tables in our database, but let's just focus on 4 tables around customer purchases.

1.0.7 From this representation you can start to see the makings of a "STAR". You will have one fact table (the center of the star) and 3 dimension tables that are coming from it.

1.0.8 TO-DO: Create the Fact table and insert the data into the table

```
In [8]: try:
        cur.execute("DROP TABLE customer_transactions;")
    except psycopg2.Error as e:
        print("Error: Issue creating table")
        print (e)

    try:
        cur.execute("CREATE TABLE IF NOT EXISTS customer_transactions (customer_id int, store_id int, spent float)")
    except psycopg2.Error as e:
        print("Error: Issue creating table")
        print (e)

    #Insert into all tables
    try:
        cur.execute("INSERT INTO customer_transactions (customer_id, store_id, spent) \
                    VALUES (%s, %s, %s)", \
                    (1, 1, 20.50))
    except psycopg2.Error as e:
        print("Error: Inserting Rows")
        print (e)

    try:
        cur.execute("INSERT INTO customer_transactions (customer_id, store_id, spent) \
                    VALUES (%s, %s, %s)", \
                    (2, 1, 35.21))
    except psycopg2.Error as e:
        print("Error: Inserting Rows")
        print (e)
```

1.0.9 TO-DO: Create the Dimension tables and insert data into those tables.

```
In [15]: try:
        cur.execute("DROP TABLE customer;")
    except psycopg2.Error as e:
        print("Error: Issue creating table")
```

```

        print (e)

try:
    cur.execute("DROP TABLE items_purchased;")
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)

try:
    cur.execute("DROP TABLE store;")
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)


try:
    cur.execute("CREATE TABLE IF NOT EXISTS customer (customer_id int, name varchar, rewards boolean);")
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)

try:
    cur.execute("INSERT INTO customer (customer_id, name, rewards) \
                VALUES(%s, %s, %s)", \
                (1, "Amanda", True))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO customer (customer_id, name, rewards) \
                VALUES(%s, %s, %s)", \
                (2, "Toby", False))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("CREATE TABLE IF NOT EXISTS items_purchased (customer_id int, item_number int, item_name varchar);")
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)

try:
    cur.execute("INSERT INTO items_purchased (customer_id, item_number, item_name) \
                VALUES(%s, %s, %s)", \
                (1, 1, "Milk"))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

```

```

        (1, 1, "Rubber Soul"))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)
try:
    cur.execute("INSERT INTO items_purchased (customer_id, item_number, item_name) \
                VALUES(%s, %s, %s)", \
                (2, 3, "Let It Be"))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("CREATE TABLE IF NOT EXISTS store (store_id int, state varchar);")
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)

try:
    cur.execute("INSERT INTO store (store_id, state) \
                VALUES(%s, %s )", \
                (1, "CA"))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO store (store_id, state) \
                VALUES(%s, %s )", \
                (2, "WA"))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

```

1.0.10 Now run the following queries on this data easily because of utilizing the Fact/ Dimension and Star Schema

Query 1: Find all the customers that spent more than 30 dollars, who are they, which store they bought it from, location of the store, what they bought and if they are a rewards member.

Query 2: How much did Customer 2 spend?

1.0.11 Query 1:

```

In [19]: try:
        cur.execute("SELECT customer.name, store.store_id, store.state, items_purchased.item_name,
                    FROM (((customer_transactions \
                    JOIN customer ON customer_transactions.customer_id = customer.customer_id)

```

```
JOIN store ON customer_transactions.store_id = store.store_id) \
JOIN items_purchased ON items_purchased.customer_id = customer_transactions.customer_id
WHERE customer_transactions.spent > 30;")
```

```
except psycopg2.Error as e:
    print("Error: select *")
    print (e)

row = cur.fetchone()
while row:
    print(row)
    row = cur.fetchone()

('Toby', 1, 'CA', 'Let It Be', False)
```

1.0.12 Your output from the above cell should look like this:

('Toby', 1, 'CA', 'Let It Be', False)

1.0.13 Query 2:

```
In [22]: try:
        cur.execute("SELECT customer_id, SUM(spent) from customer_transactions \
                    WHERE customer_id = 2 \
                    GROUP BY customer_id;")

except psycopg2.Error as e:
    print("Error: select *")
    print (e)

row = cur.fetchone()
while row:
    print(row)
    row = cur.fetchone()

(2, Decimal('35.21'))
```

1.0.14 Your output from the above cell should include Customer 2 and the amount:

(2, 35.21)

1.0.15 Summary: You can see here from this elegant schema that we were: 1) able to get "facts/metrics" from our fact table (how much each store sold), and 2) information about our customers that will allow us to do more indepth analytics to get answers to business questions by utilizing our fact and dimension tables.

1.0.16 TO-DO: Drop the tables

```
In [23]: try:
        cur.execute("DROP TABLE customer;")
    except psycopg2.Error as e:
        print("Error: Issue creating table")
        print (e)

    try:
        cur.execute("DROP TABLE items_purchased;")
    except psycopg2.Error as e:
        print("Error: Issue creating table")
        print (e)

    try:
        cur.execute("DROP TABLE store;")
    except psycopg2.Error as e:
        print("Error: Issue creating table")
        print (e)

    try:
        cur.execute("DROP TABLE customer_transactions;")
    except psycopg2.Error as e:
        print("Error: Issue creating table")
        print (e)
```

1.0.17 And finally close your cursor and connection.

```
In [24]: cur.close()
        conn.close()
```

```
In [ ]:
```