Lesson 2 Exercise 1 Creating Normalized Tables

January 18, 2023

1 Lesson 2 Exercise 1: Creating Normalized Tables

1.1 In this exercise we are going to walk through the basics of modeling data in normalized form. We will create tables in PostgreSQL, insert rows of data, and do simple JOIN SQL queries to show how these mutliple tables can work together.

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

Import the library Note: An error might popup after this command has exectuted. If it does, read it carefully before ignoring.

```
In [3]: import psycopg2
```

Create a connection to the database, get a cursor, and set autocommit to true)

```
In [4]: try:
            conn = psycopg2.connect("host=127.0.0.1 dbname=studentdb user=student password=stude
        except psycopg2.Error as e:
            print("Error: Could not make connection to the Postgres database")
            print(e)
        try:
            cur = conn.cursor()
        except psycopg2.Error as e:
            print("Error: Could not get cursor to the Database")
            print(e)
        conn.set_session(autocommit=True)
```

Let's imagine we have a table called Music Store. `Table Name: music_store column 0: Transaction Id column 1: Customer Name column 2: Cashier Name column 3: Year column 4: Albums Purchased`

1.2 Now to translate this information into a CREATE Table Statement and insert the data

```
In [17]: # TO-DO: Add the CREATE Table Statement and INSERT statements to add the data in the ta

             try:
```

```
        cur.execute("DROP TABLE IF EXISTS music_store")
    except psycopg2.Error as e:
        print("Error: Issue dropping table")
        print (e)

    try:
        cur.execute("CREATE TABLE IF NOT EXISTS music_store (transaction_id int, customer_n
    except psycopg2.Error as e:
        print("Error: Issue creating table")
        print (e)

    try:
        cur.execute("INSERT INTO music_store (transaction_id, customer_name, cashier_name,
                    VALUES (%s, %s, %s, %s, %s)", \
                    (1, "Amanda", "Sam", 2000, ['Rubber Soul', 'Let it Be']))
    except psycopg2.Error as e:
        print("Error: Inserting Rows")
        print (e)

    try:
        cur.execute("INSERT INTO music_store (transaction_id, customer_name, cashier_name,
                    VALUES (%s, %s, %s, %s, %s)", \
                    (2, "Toby", "Sam", 2000, ['My Generation']))
    except psycopg2.Error as e:
        print("Error: Inserting Rows")
        print (e)

    try:
        cur.execute("INSERT INTO music_store (transaction_id, customer_name, cashier_name,
                    VALUES (%s, %s, %s, %s, %s)", \
                    (3, "Max", "Bob", 2018, ['Meet the Beatles', 'Help!']))
    except psycopg2.Error as e:
        print("Error: Inserting Rows")
        print (e)


    try:
        cur.execute("SELECT * FROM music_store;")
    except psycopg2.Error as e:
        print("Error: select *")
        print (e)

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

```
(1, 'Amanda', 'Sam', 2000, ['Rubber Soul', 'Let it Be'])
```

```
(2, 'Toby', 'Sam', 2000, ['My Generation'])
(3, 'Max', 'Bob', 2018, ['Meet the Beatles', 'Help!'])
```

Moving to 1st Normal Form (1NF)

1.2.1 TO-DO: This data has not been normalized. To get this data into 1st normal form, you need to remove any collections or list of data and break up the list of songs into individual rows.

In [18]:
```
## TO-DO: Complete the CREATE table statements and INSERT statements

try:
    cur.execute("DROP TABLE IF EXISTS music_store2")
except psycopg2.Error as e:
    print("Error: Issue dropping table")
    print (e)

try:
    cur.execute("CREATE TABLE IF NOT EXISTS music_store2 (transaction_id int, customer_
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)

try:
    cur.execute("INSERT INTO music_store2 (transaction_id, customer_name, cashier_name,
                 VALUES (%s, %s, %s, %s, %s)", \
                 (1, "Amanda", "Sam", 2000, "Rubber Soul"))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO music_store2 (transaction_id, customer_name, cashier_name,
                 VALUES (%s, %s, %s, %s, %s)", \
                 (1, "Amanda", "Sam", 2000, "Let it Be"))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO music_store2 (transaction_id, customer_name, cashier_name,
                 VALUES (%s, %s, %s, %s, %s)", \
                 (2, "Toby", "Sam", 2000, "My Generation"))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)
```

```
        try:
            cur.execute("INSERT INTO music_store2 (transaction_id, customer_name, cashier_name,
                        VALUES (%s, %s, %s, %s, %s)", \
                        (3, "Max", "Bob", 2018, "Meet the Beatles"))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)

        try:
            cur.execute("INSERT INTO music_store2 (transaction_id, customer_name, cashier_name,
                        VALUES (%s, %s, %s, %s, %s)", \
                        (3, "Max", "Bob", 2018, "Help!"))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)

        try:
            cur.execute("SELECT * FROM music_store2;")
        except psycopg2.Error as e:
            print("Error: select *")
            print (e)

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

```
(1, 'Amanda', 'Sam', 2000, 'Rubber Soul')
(1, 'Amanda', 'Sam', 2000, 'Let it Be')
(2, 'Toby', 'Sam', 2000, 'My Generation')
(3, 'Max', 'Bob', 2018, 'Meet the Beatles')
(3, 'Max', 'Bob', 2018, 'Help!')
```

Moving to 2nd Normal Form (2NF) You have now moved the data into 1NF, which is the first step in moving to 2nd Normal Form. The table is not yet in 2nd Normal Form. While each of the records in the table is unique, our Primary key (transaction id) is not unique.

1.2.2 TO-DO: Break up the table into two tables, transactions and albums sold.

```
In [19]: try:
             cur.execute("DROP TABLE IF EXISTS transactions")
         except psycopg2.Error as e:
             print("Error: Issue dropping table")
             print (e)

         try:
             cur.execute("DROP TABLE IF EXISTS albums_sold")
```

```
except psycopg2.Error as e:
    print("Error: Issue dropping table")
    print (e)

try:
    cur.execute("CREATE TABLE IF NOT EXISTS transactions (transaction_id int, customer_
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)

try:
    cur.execute("CREATE TABLE IF NOT EXISTS albums_sold (albums_id int, transaction_id
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)

try:
    cur.execute("INSERT INTO transactions (transaction_id, customer_name, cashier_name,
                VALUES (%s, %s, %s, %s)", \
                (1, "Amanda", "Sam", 2000))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO transactions (transaction_id, customer_name, cashier_name,
                VALUES (%s, %s, %s, %s)", \
                (2, "Toby", "Sam",2000))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO transactions (transaction_id, customer_name, cashier_name,
                VALUES (%s, %s, %s, %s)", \
                (3, "Max", "Bob", 2018))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO albums_sold (albums_id, transaction_id, albums_purchased)
                VALUES (%s, %s, %s)", \
                (1, 1, 'Rubber Soul'))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)
```

```
try:
    cur.execute("INSERT INTO albums_sold (albums_id, transaction_id, albums_purchased)
                 VALUES (%s, %s, %s)", \
                 (2, 1, 'Let it Be'))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO albums_sold (albums_id, transaction_id, albums_purchased)
                 VALUES (%s, %s, %s)", \
                 (3, 2, 'My Generation'))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO albums_sold (albums_id, transaction_id, albums_purchased)
                 VALUES (%s, %s, %s)", \
                 (4, 3, 'Meet the Beatles'))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO albums_sold (albums_id, transaction_id, albums_purchased)
                 VALUES (%s, %s, %s)", \
                 (5, 3, 'Help!'))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

print("Table: transactions\n")
try:
    cur.execute("SELECT * FROM transactions;")
except psycopg2.Error as e:
    print("Error: select *")
    print (e)

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

print("\nTable: albums_sold\n")
try:
    cur.execute("SELECT * FROM albums_sold;")
except psycopg2.Error as e:
```

```
            print("Error: select *")
            print (e)
        row = cur.fetchone()
        while row:
           print(row)
           row = cur.fetchone()
```

```
Table: transactions

(1, 'Amanda', 'Sam', 2000)
(2, 'Toby', 'Sam', 2000)
(3, 'Max', 'Bob', 2018)

Table: albums_sold

(1, 1, 'Rubber Soul')
(2, 1, 'Let it Be')
(3, 2, 'My Generation')
(4, 3, 'Meet the Beatles')
(5, 3, 'Help!')
```

1.2.3 TO-DO: Do a `JOIN` on these tables to get all the information in the original first Table.

```
In [20]: ## TO-DO: Complete the join on the transactions and album_sold tables

        try:
            cur.execute("SELECT * FROM transactions JOIN albums_sold ON transactions.transactic
        except psycopg2.Error as e:
            print("Error: select *")
            print (e)

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

```
(1, 'Amanda', 'Sam', 2000, 1, 1, 'Rubber Soul')
(1, 'Amanda', 'Sam', 2000, 2, 1, 'Let it Be')
(2, 'Toby', 'Sam', 2000, 3, 2, 'My Generation')
(3, 'Max', 'Bob', 2018, 4, 3, 'Meet the Beatles')
(3, 'Max', 'Bob', 2018, 5, 3, 'Help!')
```

Moving to 3rd Normal Form (3NF) Check our table for any transitive dependencies. *HINT:* Check the table for any transitive dependencies. *Transactions* can remove *Cashier Name* to its own table, called *Employees*, which will leave us with 3 tables.

1.2.4 TO-DO: Create the third table named *employees* to move to 3rd NF.

In [24]:
```
try:
    cur.execute("DROP TABLE IF EXISTS transactions2")
except psycopg2.Error as e:
    print("Error: Issue dropping table")
    print (e)

try:
    cur.execute("DROP TABLE IF EXISTS employees")
except psycopg2.Error as e:
    print("Error: Issue dropping table")
    print (e)


try:
    cur.execute("CREATE TABLE IF NOT EXISTS transactions2 (transaction_id int, customer
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)

try:
    cur.execute("CREATE TABLE IF NOT EXISTS employees (employee_id int, employee_name v
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)

try:
    cur.execute("INSERT INTO transactions2 (transaction_id, customer_name, employee_id,
                VALUES (%s, %s, %s, %s)", \
                (1, 'Amanda', 1, 2000))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO transactions2 (transaction_id, customer_name, employee_id,
                VALUES (%s, %s, %s, %s)", \
                (2, "Toby", 1,2000))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO transactions2 (transaction_id, customer_name, employee_id,
                VALUES (%s, %s, %s, %s)", \
                (3, "Max", 2, 2018))
except psycopg2.Error as e:
```

```
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO employees (employee_id, employee_name) \
                 VALUES (%s, %s)", \
                 (1, "Sam"))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

try:
    cur.execute("INSERT INTO employees (employee_id, employee_name ) \
                 VALUES (%s, %s)", \
                 (2, "Bob"))
except psycopg2.Error as e:
    print("Error: Inserting Rows")
    print (e)

print("Table: transactions2\n")
try:
    cur.execute("SELECT * FROM transactions2;")
except psycopg2.Error as e:
    print("Error: select *")
    print (e)

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

print("\nTable: albums_sold\n")
try:
    cur.execute("SELECT * FROM albums_sold;")
except psycopg2.Error as e:
    print("Error: select *")
    print (e)

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

print("\nTable: employees\n")
try:
    cur.execute("SELECT * FROM employees;")
except psycopg2.Error as e:
    print("Error: select *")
```

```
            print (e)

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

```
Table: transactions2

(1, 'Amanda', 1, 2000)
(2, 'Toby', 1, 2000)
(3, 'Max', 2, 2018)

Table: albums_sold

(1, 1, 'Rubber Soul')
(2, 1, 'Let it Be')
(3, 2, 'My Generation')
(4, 3, 'Meet the Beatles')
(5, 3, 'Help!')

Table: employees

(1, 'Sam')
(2, 'Bob')
```

1.2.5 TO-DO: Complete the last two `JOIN` on these 3 tables so we can get all the information we had in our first Table.

```
In [25]: try:
             cur.execute("SELECT * FROM (transactions2 JOIN albums_sold ON \
                                         transactions2.transaction_id = albums_sold.transaction_i
                                         employees ON transactions2.employee_id = employees.emplo
         except psycopg2.Error as e:
             print("Error: select *")
             print (e)

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

```
(1, 'Amanda', 1, 2000, 1, 1, 'Rubber Soul', 1, 'Sam')
(1, 'Amanda', 1, 2000, 2, 1, 'Let it Be', 1, 'Sam')
(2, 'Toby', 1, 2000, 3, 2, 'My Generation', 1, 'Sam')
(3, 'Max', 2, 2018, 4, 3, 'Meet the Beatles', 2, 'Bob')
(3, 'Max', 2, 2018, 5, 3, 'Help!', 2, 'Bob')
```

1.2.6 Your output for the above cell should be:

(1, 'Amanda', 1, 2000, 1, 1, 'Rubber Soul', 1, 'Sam') (1, 'Amanda', 1, 2000, 2, 1, 'Let it Be', 1, 'Sam') (2, 'Toby', 1, 2000, 3, 2, 'My Generation', 1, 'Sam') (3, 'Max', 2, 2018, 4, 3, 'Meet the Beatles', 2, 'Bob') (3, 'Max', 2, 2018, 5, 3, 'Help!', 2, 'Bob')

1.2.7 Awesome work!! You have Normalized the dataset!

1.2.8 And finally close your cursor and connection.

```
In [27]: try:
             cur.execute("DROP table music_store")
         except psycopg2.Error as e:
             print("Error: Dropping table")
             print (e)
         try:
             cur.execute("DROP table music_store2")
         except psycopg2.Error as e:
             print("Error: Dropping table")
             print (e)
         try:
             cur.execute("DROP table albums_sold")
         except psycopg2.Error as e:
             print("Error: Dropping table")
             print (e)
         try:
             cur.execute("DROP table employees")
         except psycopg2.Error as e:
             print("Error: Dropping table")
             print (e)
         try:
             cur.execute("DROP table transactions")
         except psycopg2.Error as e:
             print("Error: Dropping table")
             print (e)
         try:
             cur.execute("DROP table transactions2")
         except psycopg2.Error as e:
             print("Error: Dropping table")
             print (e)

Error: Dropping table
cursor already closed
Error: Dropping table
cursor already closed
Error: Dropping table
cursor already closed
Error: Dropping table
cursor already closed
```

```
Error: Dropping table
cursor already closed
Error: Dropping table
cursor already closed
```

1.2.9 And finally close your cursor and connection.

```
In [26]: cur.close()
         conn.close()
```

```
In [ ]:
```