Lesson 2 Exercise 2 Creating Denormalized Tables

January 6, 2023

1 Lesson 2 Exercise 2: Creating Denormalized Tables

1.1 Walk through the basics of modeling data from normalized from to denormalized form. We will create tables in PostgreSQL, insert rows of data, and do simple JOIN SQL queries to show how these multiple tables can work together.

Where you see ##### you will need to fill in code. This exercise will be more challenging than the last. Use the information provided to create the tables and write the insert statements.

Remember the examples shown are simple, but imagine these situations at scale with large datasets, many users, and the need for quick response time.

1.1.1 Import the library

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

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

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

```
In [ ]: 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 start with our normalized (3NF) database set of tables we had in the last exercise, but we have added a new table `sales`. `Table Name: transactions2 column 0: transaction Id column 1: Customer Name column 2: Cashier Id column 3: Year`

```
   Table Name: albums_sold column 0: Album Id column 1: Transaction Id column 3:
Album Name
   Table Name: employees column 0: Employee Id column 1: Employee Name
   Table Name: sales column 0: Transaction Id column 1: Amount Spent
```

1.1.3 TO-DO: Add all Create statements for all Tables and Insert data into the tables

```
In [ ]: # TO-DO: Add all Create statements for all tables
        try:
            cur.execute("CREATE TABLE transaction2 IF NOT EXISTS (transaction_id int, customer_n
        except psycopg2.Error as e:
            print("Error: Issue creating table")
            print (e)

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

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

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


        # TO-DO: Insert data into the tables




        try:
            cur.execute("INSERT INTO transactions2 (transaction_id, customer_name, cashier_id, y
                         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, cashier_id, y
                         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, cashier_id, y
                 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 albums_sold (album_id, transaction_id, album_name) \
                 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 (album_id, transaction_id, album_name) \
                 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 (album_id, transaction_id, album_name) \
                 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 (album_id, transaction_id, album_name) \
                 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 (album_id, transaction_id, album_name) \
                 VALUES (%s, %s, %s)", \
                  (5, 3, "Help!"))
    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)

    try:
        cur.execute("INSERT INTO sales (transaction_id, amount_spent) \
                     VALUES (%s, %s)", \
                     (1, 40))
    except psycopg2.Error as e:
        print("Error: Inserting Rows")
        print (e)

    try:
        cur.execute("INSERT INTO sales (transaction_id, amount_spent) \
                     VALUES (%s, %s)", \
                     (2, 19))
    except psycopg2.Error as e:
        print("Error: Inserting Rows")
        print (e)

    try:
        cur.execute("INSERT INTO sales (transaction_id, amount_spent) \
                     VALUES (%s, %s)", \
                     (3, 45))
    except psycopg2.Error as e:
        print("Error: Inserting Rows")
        print (e)
```

TO-DO: Confirm using the Select statement the data were added correctly

```
In [ ]: 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()

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

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

1.1.4 Let's say you need to do a query that gives:

`transaction_id  customer_name  cashier name  year   albums sold  amount sold`

1.1.5 TO-DO: Complete the statement below to perform a 3 way `JOIN` on the 4 tables you have created.

```
In [ ]: try:
            cur.execute("SELECT * FROM transactions2 t JOIN albums_sold a ON t.transaction_id =


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

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

Great we were able to get the data we wanted.

1.1.6 But, we had to perform a 3 way `JOIN` to get there. While it's great we had that flexibility, we need to remember that `JOINS` are slow and if we have a read heavy workload that required low latency queries we want to reduce the number of `JOINS`. Let's think about denormalizing our normalized tables.

1.1.7 With denormalization you want to think about the queries you are running and how to reduce the number of JOINS even if that means duplicating data. The following are the queries you need to run.

Query 1 : `select transaction_id, customer_name, amount_spent FROM <min number of tables>` It should generate the amount spent on each transaction #### Query 2: `select cashier_name, SUM(amount_spent) FROM <min number of tables> GROUP BY cashier_name` It should generate the total sales by cashier

1.1.8 Query 1: `select transaction_id, customer_name, amount_spent FROM <min number of tables>`

One way to do this would be to do a JOIN on the `sales` and `transactions2` table but we want to minimize the use of `JOINS`.

To reduce the number of tables, first add `amount_spent` to the `transactions` table so that you will not need to do a JOIN at all.

`Table Name: transactions  column 0: transaction Id column 1: Customer Name column 2: Cashier Id column 3: Year column 4: amount_spent`

1.1.9 TO-DO: Add the tables as part of the denormalization process

```
In [ ]: # TO-DO: Create all tables
        try:
            cur.execute("#####")
        except psycopg2.Error as e:
            print("Error: Issue creating table")
            print (e)



        #Insert data into all tables

        try:
            cur.execute("INSERT INTO transactions (#####) \
                         VALUES (%s, %s, %s, %s, %s)", \
                         (#####))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)

        try:
            cur.execute("INSERT INTO transactions (#####) \
                         VALUES (%s, %s, %s, %s, %s)", \
                         (#####))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)

        try:
            cur.execute("INSERT INTO transactions (#####) \
                         VALUES (%s, %s, %s, %s, %s)", \
                         (#####))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)
```

1.1.10 Now you should be able to do a simplifed query to get the information you need. No `JOIN` is needed.

```
In [ ]: try:
            cur.execute("#####")

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

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

Your output for the above cell should be the following: (1, 'Amanda', 40) (2, 'Toby', 19) (3, 'Max', 45)

1.1.11 Query 2: `select cashier_name, SUM(amount_spent) FROM <min number of tables> GROUP BY cashier_name`

To avoid using any `JOINS`, first create a new table with just the information we need.
`Table Name: cashier_sales col: Transaction Id Col: Cashier Name Col: Cashier Id col: Amount_Spent`

1.1.12 TO-DO: Create a new table with just the information you need.

```
In [ ]: # Create the tables

        try:
            cur.execute("#####")
        except psycopg2.Error as e:
            print("Error: Issue creating table")
            print (e)


        #Insert into all tables

        try:
            cur.execute("INSERT INTO ##### (#####) \
                         VALUES (%s, %s, %s, %s)", \
                         (##### ))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)

        try:
            cur.execute("INSERT INTO ##### (#####) \
                         VALUES (%s, %s, %s, %s)", \
                         (##### ))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)

        try:
            cur.execute("INSERT INTO ##### (#####) \
                         VALUES (%s, %s, %s, %s)", \
                         (#####))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)
```

1.1.13 Run the query

```
In [ ]: try:
            cur.execute("#####")

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

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

Your output for the above cell should be the following: ('Sam', 59) ('Bob', 45)

We have successfully taken normalized table and denormalized them inorder to speed up our performance and allow for simplier queries to be executed.

1.1.14 Drop the tables

```
In [ ]: try:
            cur.execute("DROP table ####")
        except psycopg2.Error as e:
            print("Error: Dropping table")
            print (e)
        try:
            cur.execute("DROP table #####")
        except psycopg2.Error as e:
            print("Error: Dropping table")
            print (e)
        try:
            cur.execute("DROP table #####")
        except psycopg2.Error as e:
            print("Error: Dropping table")
            print (e)
        try:
            cur.execute("DROP table #####")
        except psycopg2.Error as e:
            print("Error: Dropping table")
            print (e)
        try:
            cur.execute("DROP table #####")
        except psycopg2.Error as e:
            print("Error: Dropping table")
            print (e)
        try:
            cur.execute("DROP table #####")
        except psycopg2.Error as e:
            print("Error: Dropping table")
            print (e)
```

1.1.15 And finally close your cursor and connection.

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