Lesson 2 Exercise 2 Creating Denormalized Tables–ANSWER KEY

January 21, 2023

1 Lesson 2 Exercise 2 Solution: Creating Denormalized Tables

1.0.1 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 mutliple tables can work together.

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

1.0.2 Import the library

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

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

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

```
In [2]: 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`

1.0.4 We add CREATE statements for all tables and INSERT data into the tables

In [3]:
```
try:
    cur.execute("CREATE TABLE IF NOT EXISTS transactions2 (transaction_id int, \
                                                           customer_name varchar, cashie
                                                           year int);")
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 varchar);")
except psycopg2.Error as e:
    print("Error: Issue creating table")
    print (e)

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

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


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)
```

1.0.5 Confirm the tables were created with the data

```
In [4]: 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()

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')

Table: Sales

(1, 40)
(2, 19)
(3, 45)
```

Let's say we need to do a query that gives us: `transaction_id customer_name cashier name year albums sold amount sold`

we will need to perform a 3 way `JOIN` on the 4 tables we have created.

```
In [5]: try:
            cur.execute("SELECT transactions2.transaction_id, customer_name, employees.employee_
                                year, albums_sold.album_name, sales.amount_spent\
                        FROM ((transactions2 JOIN employees ON \
                               transactions2.cashier_id = employees.employee_id) JOIN \
                               albums_sold ON albums_sold.transaction_id=transactions2.transac
                               sales ON transactions2.transaction_id=sales.transaction_id;")


        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', 40)
(1, 'Amanda', 'Sam', 2000, 'Let It Be', 40)
(2, 'Toby', 'Sam', 2000, 'My Generation', 19)
(3, 'Max', 'Bob', 2018, 'Meet the Beatles', 45)
(3, 'Max', 'Bob', 2018, 'Help!', 45)
```

Great we were able to get the data we wanted.

1.0.6 But, we had a to 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.

1.0.7 With denormalization we want to think about the queries we are running and how we can reduce our number of JOINS even if that means duplicating data

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

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

There are two ways to do this, you can do a JOIN on the `sales` and `transactions2` table but we want to minimize the use of `JOINS`.

Let's add `amount_spent` to the `transactions` table so that we 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`

```
In [6]: #Create all Tables and insert the data

        try:
            cur.execute("CREATE TABLE IF NOT EXISTS transactions (transaction_id int, \
                                                                  customer_name varchar, cashie
                                                                  year int, amount_spent int);"
        except psycopg2.Error as e:
            print("Error: Issue creating table")
            print (e)



        #Insert into all tables

        try:
            cur.execute("INSERT INTO transactions (transaction_id, customer_name, cashier_id, ye
                        VALUES (%s, %s, %s, %s, %s)", \
                        (1, "Amanda", 1, 2000, 40))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)

        try:
            cur.execute("INSERT INTO transactions (transaction_id, customer_name, cashier_id, ye
                        VALUES (%s, %s, %s, %s, %s)", \
```

```
                    (2, "Toby", 1, 2000, 19))
    except psycopg2.Error as e:
        print("Error: Inserting Rows")
        print (e)

    try:
        cur.execute("INSERT INTO transactions (transaction_id, customer_name, cashier_id, ye
                    VALUES (%s, %s, %s, %s, %s)", \
                    (3, "Max", 2, 2018, 45))
    except psycopg2.Error as e:
        print("Error: Inserting Rows")
        print (e)
```

1.0.9 Great we can now do a simplifed query to get the information we need. No JOIN is needed.

```
In [7]: try:
            cur.execute("SELECT transaction_id, customer_name, amount_spent FROM transactions;")

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

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

```
(1, 'Amanda', 40)
(2, 'Toby', 19)
(3, 'Max', 45)
```

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

We could also do a JOIN on the tables we have created, but what if we do not want to have any JOINS, why not 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

```
In [8]: try:
            cur.execute("CREATE TABLE IF NOT EXISTS cashier_sales (transaction_id int, cashier_n
                                                                   cashier_id int, amount_spent
        except psycopg2.Error as e:
            print("Error: Issue creating table")
            print (e)
```

```
        #Insert into all tables

        try:
            cur.execute("INSERT INTO cashier_sales (transaction_id, cashier_name, cashier_id, am
                        VALUES (%s, %s, %s, %s)", \
                        (1, "Sam", 1, 40 ))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)

        try:
            cur.execute("INSERT INTO cashier_sales (transaction_id, cashier_name, cashier_id, am
                        VALUES (%s, %s, %s, %s)", \
                        (2, "Sam", 1, 19 ))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)

        try:
            cur.execute("INSERT INTO cashier_sales (transaction_id, cashier_name, cashier_id, am
                        VALUES (%s, %s, %s, %s)", \
                        (3, "Bob", 2, 45))
        except psycopg2.Error as e:
            print("Error: Inserting Rows")
            print (e)
```

Now let's run our query

```
In [9]: try:
            cur.execute("select cashier_name, SUM(amount_spent) FROM cashier_sales GROUP BY cash

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

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

```
('Sam', 59)
('Max', 45)
```

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

1.0.11 Drop the tables.

```
In [10]: 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)
         try:
             cur.execute("DROP table sales")
         except psycopg2.Error as e:
             print("Error: Dropping table")
             print (e)
         try:
             cur.execute("DROP table cashier_sales")
         except psycopg2.Error as e:
             print("Error: Dropping table")
             print (e)
```

1.0.12 And finally close your cursor and connection.

```
In [11]: cur.close()
         conn.close()
```

```
In [ ]:
```