UNIVERSIDAD DEL VALLE DE GUATEMALA
CC3088 - Base de Datos 1
Sección 10
Ing. Héctor Antonio Hurtarte

Laboratorio 09 - Índices y Triggers

Mario Cristales, 21631
Javier Ramirez, 21600

GUATEMALA, 20 de marzo de 2023

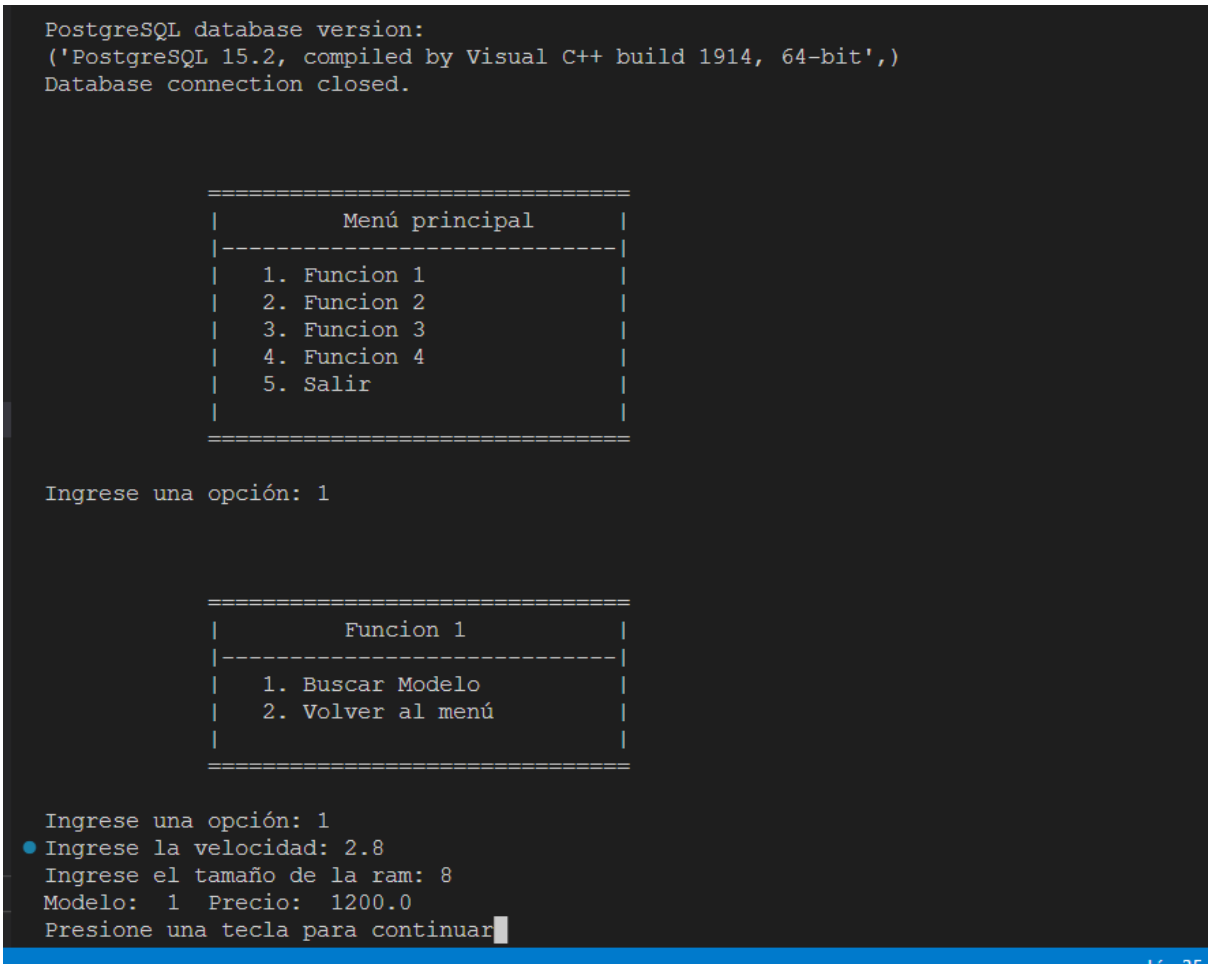
Ejercicio 1

Desarrolle el siguiente esquema relacional en una base de datos PostgreSQL, utilizando los tipos de datos que considere más convenientes:

Producto (fabricante, modelo, tipo)
PC (modelo, velocidad, ram, disco, precio)

Recuerde del Laboratorio #8 sobre Álgebra relacional que tipo puede ser uno de tres valores: “PC”, “laptop” o “impresora”, pero en este laboratorio nos interesa solo trabajar con PCs.
Desarrolle un programa en Python capaz de interactuar con su base de datos y que contenga las funciones que se describen a continuación. Investigue y agregue a sus funciones las instrucciones BEGIN TRANSACTION, COMMIT y ROLLBACK en los momentos que considere pertinentes considerando que múltiples usuarios pueden invocar sus funciones concurrentemente, y recuerde especificar a su programa las transacciones que sean de solo lectura.

Función 1: Dada una velocidad y un tamaño de RAM como argumentos de la función, localice las PCs con esa velocidad y RAM, e imprima el número de modelo y precio de cada una.



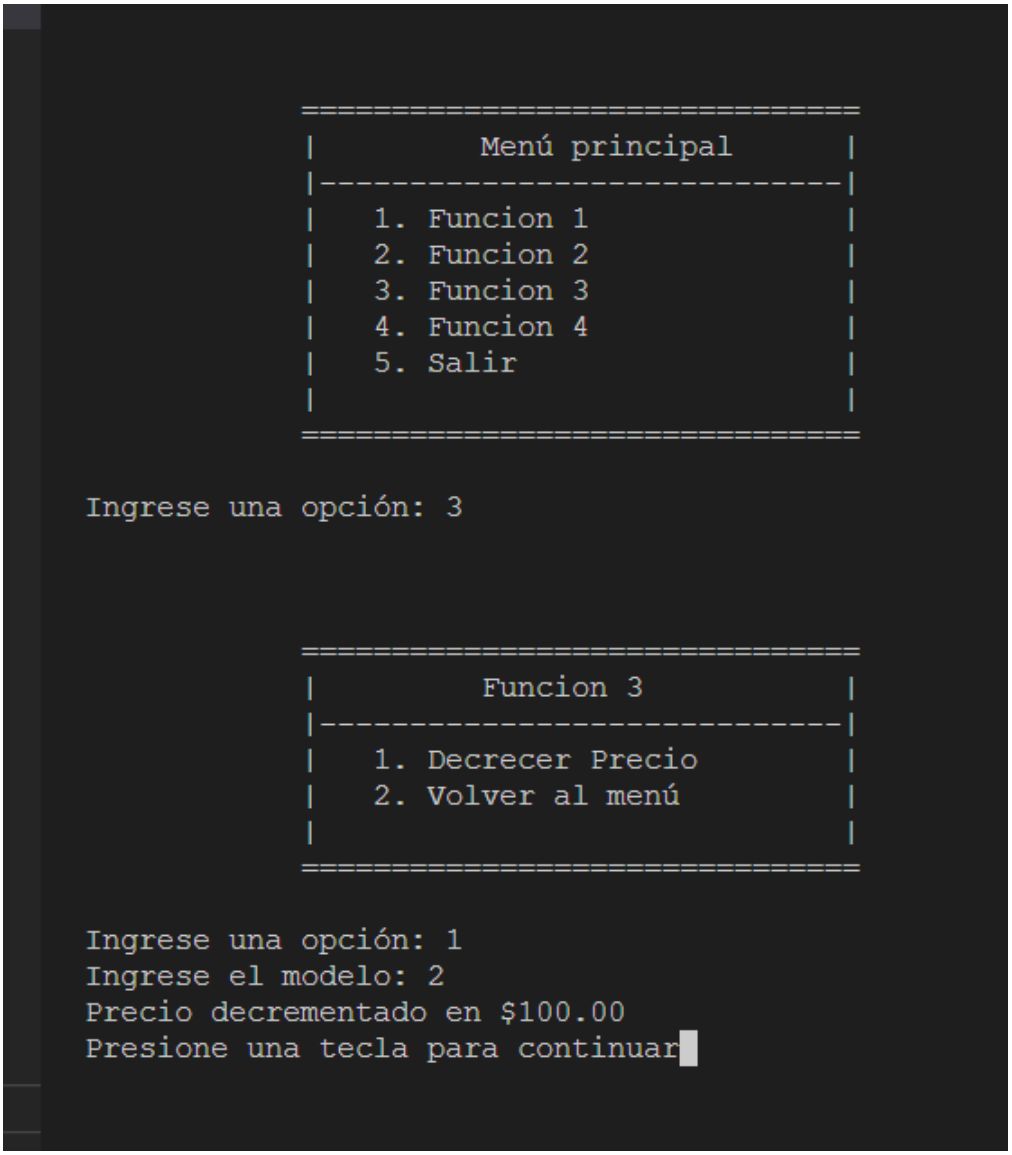
```python
def buscar_pc(velocidad, ram):
    try:
        # Configuración de la conexión a la base de datos
        params = config()
        conn = psycopg2.connect(**params)
        cur = conn.cursor(cursor_factory=RealDictCursor)

        # Comenzamos una transacción
        conn.autocommit = False

        # Buscamos las PCs con la velocidad y RAM especificadas
        cur.execute("SELECT modelo, precio FROM PC WHERE velocidad = %s::float AND ram = %s::integer", (velocidad, ram))
        pcs = cur.fetchall()

        # Hacemos un commit para confirmar los cambios
        conn.commit()

        return pcs

    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
        # Si ocurre un error, hacemos un rollback para deshacer los cambios
        conn.rollback()

    finally:
        # Cerramos la conexión a la base de datos
        if conn is not None:
            cur.close()
            conn.close()
```

Función 2: Dado un número de modelo, elimine la tupla para ese modelo tanto de la relación PC como de la relación Producto. Si el número de modelo no existe su programa debe hacerlo saber.

```
                ================================
                |         Menú principal       |
                |------------------------------|
                |   1. Funcion 1               |
                |   2. Funcion 2               |
                |   3. Funcion 3               |
                |   4. Funcion 4               |
                |   5. Salir                   |
                |                              |
                ================================

  Ingrese una opción: 2


                ================================
                |         Funcion 2            |
                |------------------------------|
                |   1. Eliminar Modelo         |
                |   2. Volver al menú          |
                |                              |
                ================================

  Ingrese una opción: 1
  Ingrese el modelo: 1
  Tupla eliminada de la tabla PC
  Tupla eliminada de la tabla Producto
```

```python
def eliminar_modelo(modelo):
    try:
        # Configuración de la conexión a la base de datos
        params = config()
        conn = psycopg2.connect(**params)
        cur = conn.cursor()

        # Comenzamos una transacción
        conn.autocommit = False

        # Verificamos si el modelo existe en la tabla PC
        cur.execute("SELECT COUNT(*) FROM PC WHERE modelo = %s", (modelo,))
        pc_count = cur.fetchone()[0]

        if pc_count > 0:
            # Si el modelo existe en la tabla PC, eliminamos la tupla correspondiente
            cur.execute("DELETE FROM PC WHERE modelo = %s", (modelo,))
            print("Tupla eliminada de la tabla PC")

        # Verificamos si el modelo existe en la tabla Producto
        cur.execute("SELECT COUNT(*) FROM Producto WHERE modelo = %s", (modelo,))
        producto_count = cur.fetchone()[0]

        if producto_count > 0:
            # Si el modelo existe en la tabla Producto, eliminamos la tupla correspondiente
            cur.execute("DELETE FROM Producto WHERE modelo = %s", (modelo,))
            print("Tupla eliminada de la tabla Producto")

        if pc_count == 0 and producto_count == 0:
            # Si el modelo no existe en ninguna de las dos tablas, imprimimos un mensaje indicando que no existe
            print("El modelo especificado no existe")

        # Hacemos un commit para confirmar los cambios
        conn.commit()

    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
        # Si ocurre un error, hacemos un rollback para deshacer los cambios
        conn.rollback("Error al eliminar la tupla")

    finally:
        # Cerramos la conexión a la base de datos
        if conn is not None:
            cur.close()
            conn.close()
```

Función 3: Dado un número de modelo, decrecer el precio de ese modelo en $100.00. Si el número de modelo no existe su programa debe hacerlo saber.



```python
def decrementar_precio(modelo):
    try:
        # Configuración de la conexión a la base de datos
        params = config()
        conn = psycopg2.connect(**params)
        cur = conn.cursor()

        # Comenzamos una transacción
        conn.autocommit = False

        # Verificamos si el modelo existe en la tabla Producto
        cur.execute("SELECT COUNT(*) FROM Producto WHERE modelo = %s", (modelo,))
        producto_count = cur.fetchone()[0]

        if producto_count > 0:
            # Si el modelo existe en la tabla Producto, decrementamos el precio en $100.00
            cur.execute("UPDATE PC SET precio = precio - 100.00 WHERE modelo = %s", (modelo,))
            print("Precio decrementado en $100.00")

        if producto_count == 0:
            # Si el modelo no existe en la tabla Producto, imprimimos un mensaje indicando que no existe
            print("El modelo especificado no existe")

        # Hacemos un commit para confirmar los cambios
        conn.commit()

    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
        # Si ocurre un error, hacemos un rollback para deshacer los cambios
        conn.rollback()

    finally:
        # Cerramos la conexión a la base de datos
        if conn is not None:
            cur.close()
            conn.close()
```

```
        ('Fabricante5','PC');

INSERT INTO PC (modelo, velocidad, ram, disco, precio)
VALUES (1, 2.8, 8, 500, 1200.00),
       (2, 3.2, 16, 1000, 1800.00),
       (3, 2.4, 4, 250, 600.00),
       (4, 2.2, 4, 1000, 900.00),
       (5, 3.0, 8, 500, 1100.00);


select * from producto ;

select * from pc;
```

pc 1

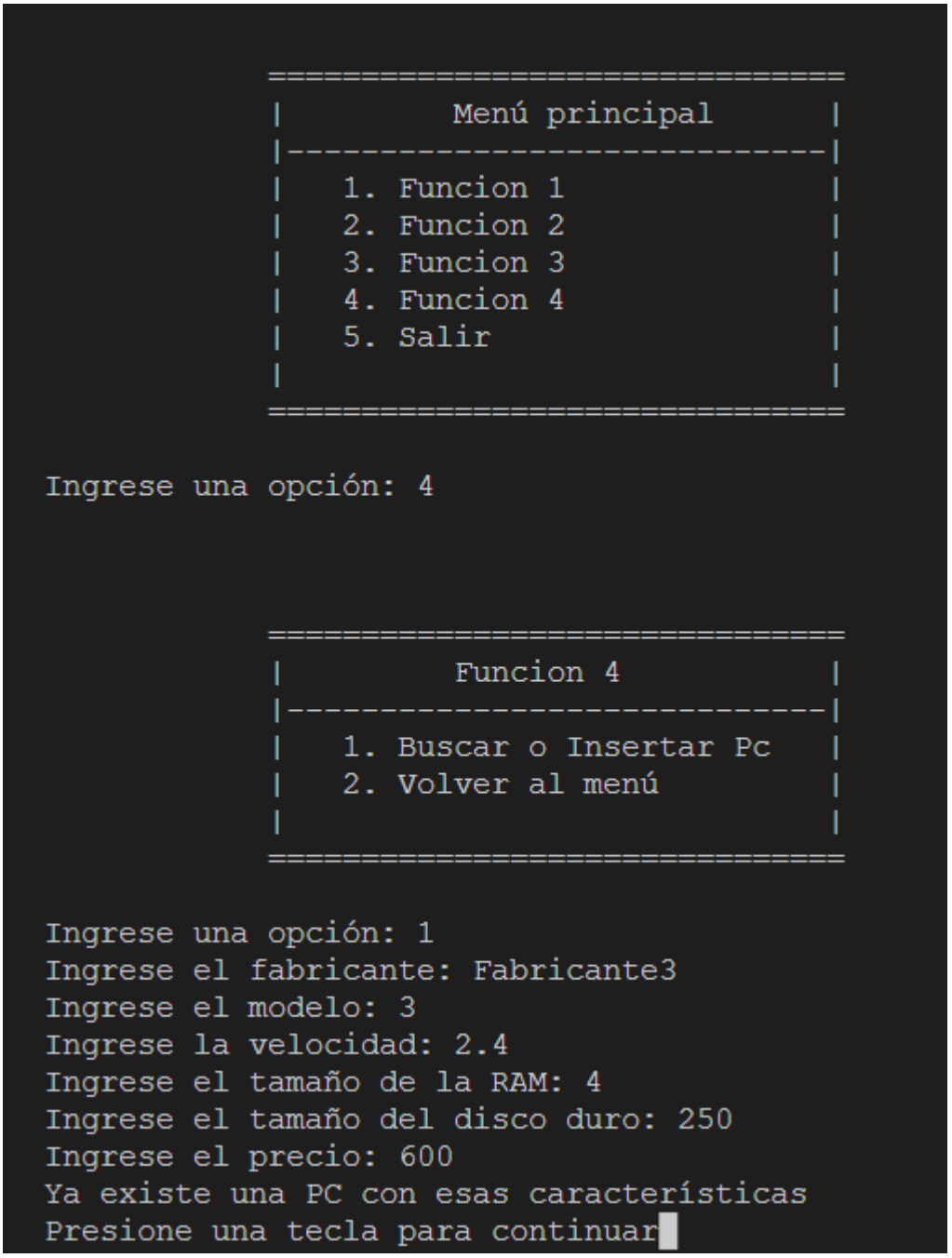
select * from pc | Enter a SQL expression to filter results (use Ctrl+Space)

Grilla | Texto

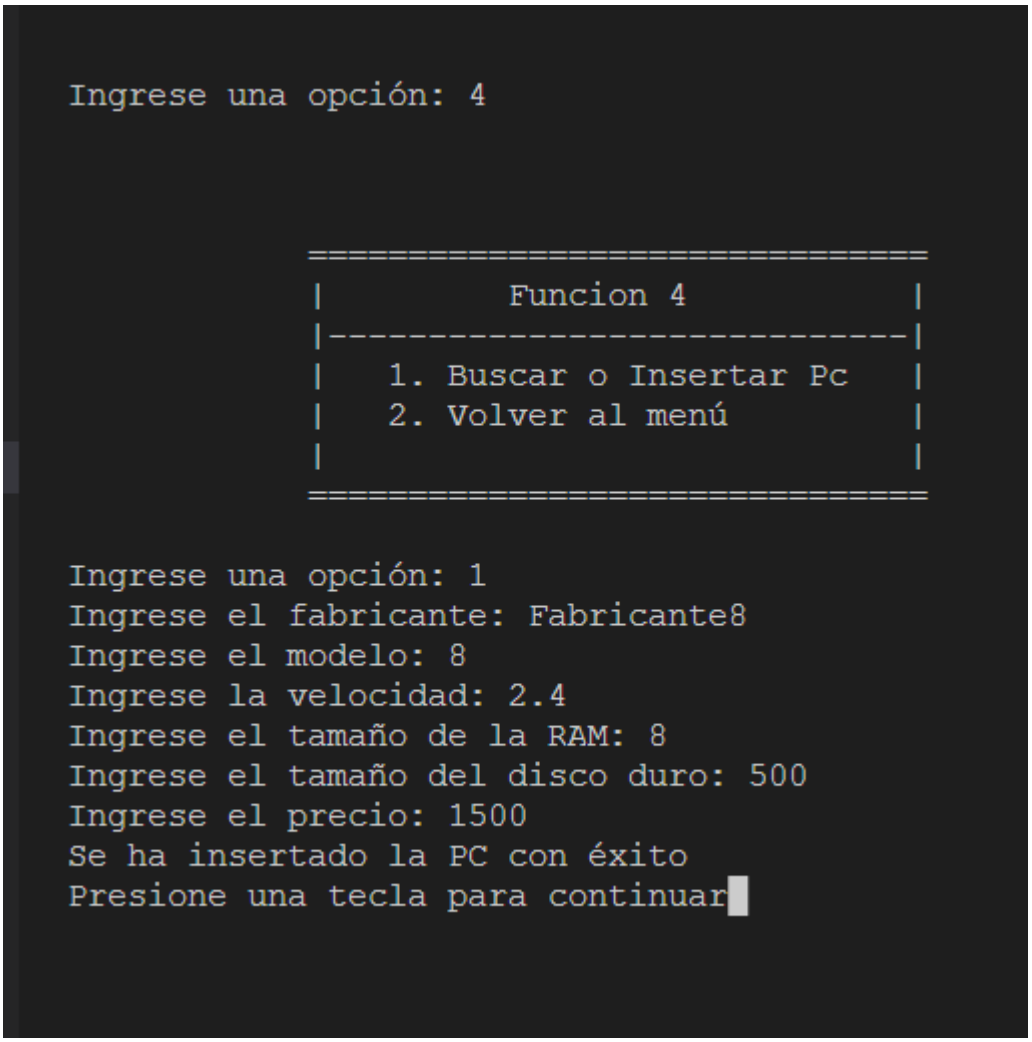
modelo	velocidad	ram	disco	precio
3	2.4	4	250	600.0
4	2.2	4	1000	900.0
5	3.0	8	500	1100.0
2	3.2	16	1000	1700.0

Función 4: Dado un fabricante, número de modelo, velocidad de procesador, tamaño de RAM tamaño de disco duro y precio verifique si hay o no una PC que cumpla con esas características.

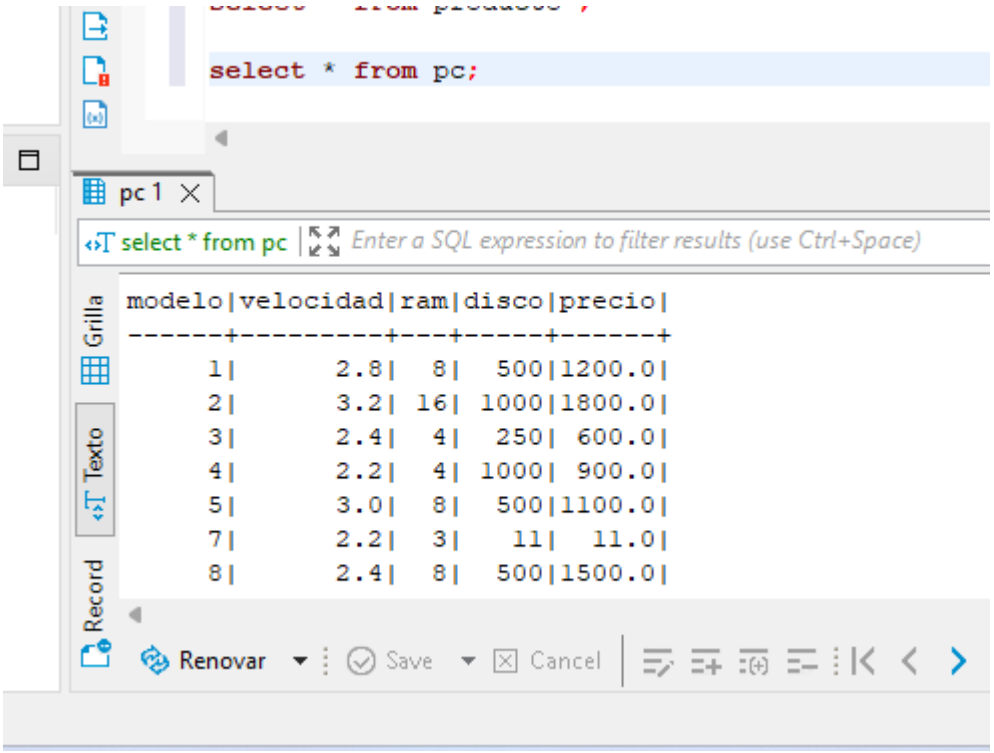
(i) Si sí encuentra un modelo muestre un mensaje de error al usuario



(ii) Si no se encuentra un modelo insértelo en las tablas Producto y PC.




```python
def buscar_o_insertar_pc(fabricante, modelo, velocidad, ram, disco, precio):
    try:
        # Configuración de la conexión a la base de datos
        params = config()
        conn = psycopg2.connect(**params)
        cur = conn.cursor()

        # Comenzamos una transacción
        conn.autocommit = False

        # Buscamos si hay una PC que cumpla con las características especificadas
        cur.execute("""SELECT COUNT(*) FROM PC p
                       JOIN Producto pr ON p.modelo = pr.modelo
                       WHERE pr.fabricante = %s AND p.modelo = %s
                       AND p.velocidad = %s AND p.ram = %s
                       AND p.disco = %s AND p.precio = %s""",
                    (fabricante, modelo, velocidad, ram, disco, precio))
        pc_count = cur.fetchone()[0]

        if pc_count > 0:
            # Si se encuentra un modelo con las características especificadas, mostramos un mensaje de error al usuario
            print("Ya existe una PC con esas características")

        else:
            # Si no se encuentra un modelo con las características especificadas, lo insertamos en las tablas Producto y PC
            cur.execute("INSERT INTO Producto(fabricante,modelo, tipo) VALUES (%s,%s, 'PC')", (fabricante,modelo))
            cur.execute("INSERT INTO PC(modelo, velocidad, ram, disco, precio) VALUES (%s, %s, %s, %s, %s)", (modelo, velocidad, ram, disco, precio))
            print("Se ha insertado la PC con éxito")

        # Hacemos un commit para confirmar los cambios
        conn.commit()

    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
        # Si ocurre un error, hacemos un rollback para deshacer los cambios
        conn.rollback()

    finally:
        # Cerramos la conexión a la base de datos
        if conn is not None:
            cur.close()
            conn.close()
```



Su programa debe tener una interfaz de línea de comando (CLI) o una interfaz gráfica (GUI) que permita interactuar correctamente con estas funciones especificando los parámetros solicitados.

Ejercicio 2: Declaración de índices

Para cada una de las funciones del ejercicio anterior discuta qué posibles problemas de atomicidad (si los hubiera) que podrían ocurrir si el sistema fuera interrumpido durante cualquier punto de la ejecución de la función. Discuta además qué posibles problemas de concurrenciapodían darse si una invocación de la función ocurre al mismo tiempo que otra. Su respuesta debe incluir el análisis para cada una de las 4 funciones.

Función 1:

Problemas de atomicidad: La función no involucra transacciones o cambios en la base de datos, por lo que no hay problemas de atomicidad si el sistema se interrumpe en cualquier punto de la ejecución de la función.

Problemas de concurrencia: Como la función solo realiza consultas a la base de datos, no hay posibles problemas de concurrencia si una invocación de la función ocurre al mismo tiempo que otra.

Función 2:

Problemas de atomicidad: La función utiliza una transacción para actualizar la tabla Producto y luego la tabla Laptop, por lo que si el sistema se interrumpe en cualquier punto de la ejecución de la función, puede haber problemas de atomicidad que dejen la base de datos en un estado inconsistente.
Problemas de concurrencia: Si varias invocaciones de la función ocurren al mismo tiempo, pueden ocurrir problemas de concurrencia si dos o más transacciones intentan actualizar las mismas filas en la tabla Producto o Laptop simultáneamente.

Función 3:

Problemas de atomicidad: La función utiliza una transacción para actualizar la tabla Producto y luego la tabla Impresora, por lo que si el sistema se interrumpe en cualquier punto de la ejecución de la función, puede haber problemas de atomicidad que dejen la base de datos en un estado inconsistente.

Problemas de concurrencia: Si varias invocaciones de la función ocurren al mismo tiempo, pueden ocurrir problemas de concurrencia si dos o más transacciones intentan actualizar las mismas filas en la tabla Producto o Impresora simultáneamente.

Función 4:

Problemas de atomicidad: La función utiliza una transacción para buscar o insertar una nueva PC en la base de datos. Si el sistema se interrumpe en cualquier punto de la ejecución de la función, puede haber problemas de atomicidad que dejen la base de datos en un estado inconsistente.

Problemas de concurrencia: Si varias invocaciones de la función ocurren al mismo tiempo, pueden ocurrir problemas de concurrencia si dos o más transacciones intentan actualizar la misma tabla simultáneamente. Además, puede haber problemas si dos o más transacciones intentan buscar o insertar la misma PC simultáneamente, lo que puede resultar en la inserción de una PC duplicada en la base de datos.

Ejercicio 3

Suponga que se ejecuta como una transacción T una de las cuatro funciones del ejercicio 1, mientras otras transacciones de esa misma función o de cualquier otra ocurren casi al mismo tiempo. ¿Qué diferencias podría observar para el resultado de T si todas las transacciones se ejecutan con un nivel de aislamiento READ UNCOMMITED en lugar de SERIALIZABLE? Considere por separado el caso en que T es cada una de las 4 funciones definidas, es decir que su respuesta debe incluir el análisis para cada una de las 4 funciones

Si todas las transacciones se ejecutan con un nivel de aislamiento READ UNCOMMITTED en lugar de SERIALIZABLE, se permitiría la lectura de datos que aún no han sido confirmados en la base de datos, lo que podría llevar a que la transacción T obtenga datos no válidos. En otras palabras, READ UNCOMMITTED permite leer datos que aún no han sido confirmados o que aún no han sido escritos en la base de datos.

Función 1: Si se ejecuta con un nivel de aislamiento READ UNCOMMITTED, puede haber un problema de concurrencia si otra transacción modifica el stock de un producto mientras la transacción T está en proceso. Esto puede resultar en que la transacción T obtenga un stock no válido para el producto en cuestión.

Función 2: Si se ejecuta con un nivel de aislamiento READ UNCOMMITTED, podría ocurrir un problema de concurrencia si otra transacción modifica el precio de venta de un producto mientras la transacción T está en proceso. Esto podría resultar en que la transacción T calcule un precio de venta no válido para el producto.

Función 3: Si se ejecuta con un nivel de aislamiento READ UNCOMMITTED, puede haber un problema de concurrencia si otra transacción modifica el número de ventas de un producto mientras la transacción T está en proceso. Esto podría resultar en que la transacción T calcule un número de ventas no válido para el producto.

Función 4: Si se ejecuta con un nivel de aislamiento READ UNCOMMITTED, podría haber un problema de concurrencia si otra transacción inserta una PC con las mismas características que la que se está insertando en la transacción T mientras está en proceso. Esto podría resultar en que la transacción T inserte una PC duplicada en la base de datos.

Script SQL

```sql
CREATE TABLE Producto (
 fabricante VARCHAR(50),
 modelo INTEGER PRIMARY KEY,
 tipo VARCHAR(50)
);
CREATE TABLE PC (
 modelo INTEGER PRIMARY KEY,
 velocidad FLOAT,
 ram INTEGER,
 disco INTEGER,
 precio FLOAT,
 FOREIGN KEY (modelo) REFERENCES Producto(modelo)
);
INSERT INTO Producto (fabricante,modelo, tipo)
VALUES ('Fabricante1',1,'PC'),
       ('Fabricante2',2,'PC'),
       ('Fabricante3',3,'PC'),
       ('Fabricante4',4,'PC'),
       ('Fabricante5',5,'PC');
INSERT INTO PC (modelo, velocidad, ram, disco, precio)
VALUES (1, 2.8, 8, 500, 1200.00),
       (2, 3.2, 16, 1000, 1800.00),
       (3, 2.4, 4, 250, 600.00),
       (4, 2.2, 4, 1000, 900.00),
       (5, 3.0, 8, 500, 1100.00);

select * from producto ;
select * from pc;
```