УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Рефакторинг баз данных и приложений»

Этап №1

Рефакторинг существующего проекта

Выполнил:

Волненко Д.А.

Преподаватель

Логинов И. П.

Санкт-Петербург, 2024 г.

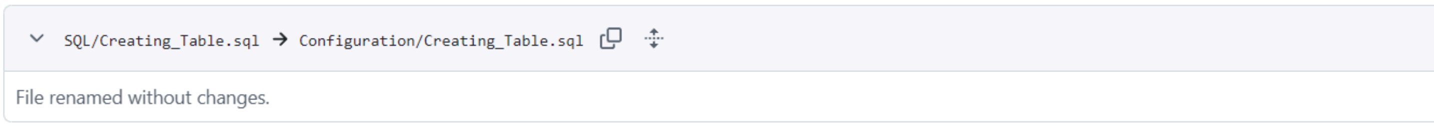
Задание

Провести рефакторинг приложения, в качестве такого был взят курсовой проект по ИСБД – telegram-бот для управления библиотекой. Для первого этапа было запланировано сделать:

1. Реструктуризацию конфигурационных файлов в единую директорию
2. Рефакторинг повторяющихся блоков кода:

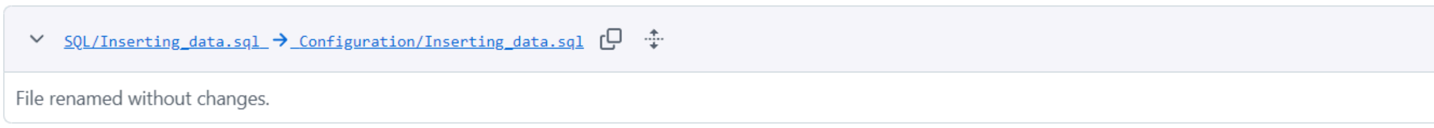
- Во многих функциях создается подключение к базе данных с помощью create_connection(), а затем используется with conn и with conn.cursor() для выполнения SQL-запросов. Вместо этого добавить общую функцию execute_query;
- Во многих функциях дублируется проверка на существование записи перед добавлением новой записи. Вместо этого добавить общую функцию check_if_exists;
- Создать декоратор для обработки общих ошибок и логирования.

Реструктуризация конфигурационных файлов в единую директорию



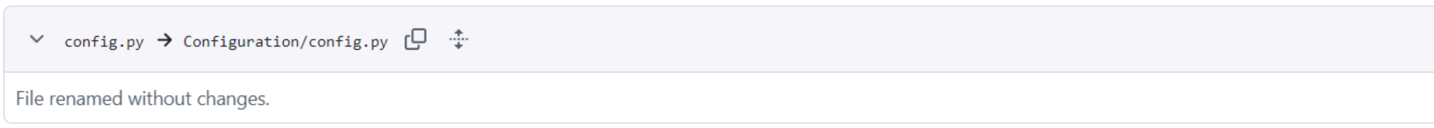
```
SQL/Creating_Table.sql → Configuration/Creating_Table.sql
```

File renamed without changes.



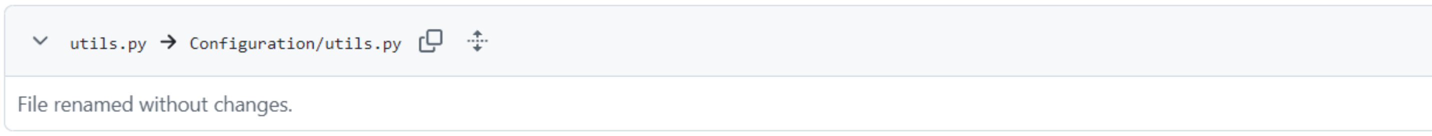
```
SQL/Inserting_data.sql → Configuration/Inserting_data.sql
```

File renamed without changes.



```
config.py → Configuration/config.py
```

File renamed without changes.



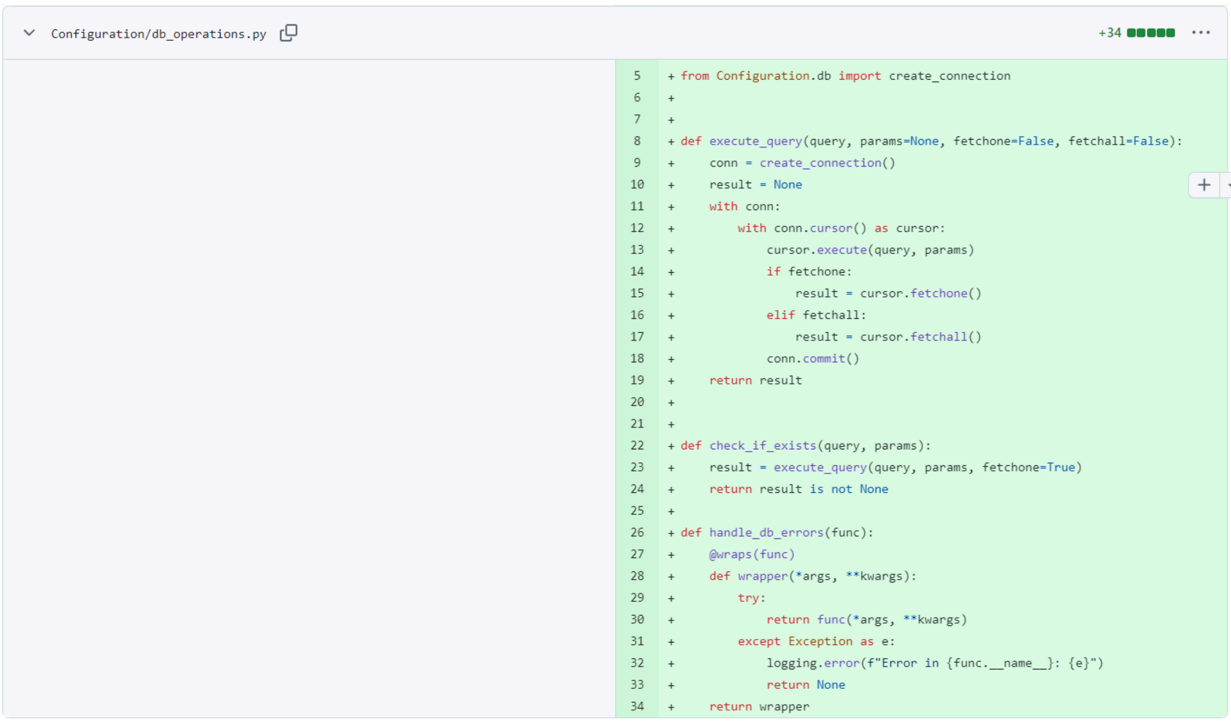
```
utils.py → Configuration/utils.py
```

File renamed without changes.

Таким образом, была создана директория Configuration, в которую переместились все файлы, связанные с базой данных, настройками бота и другими внешним зависимостями.

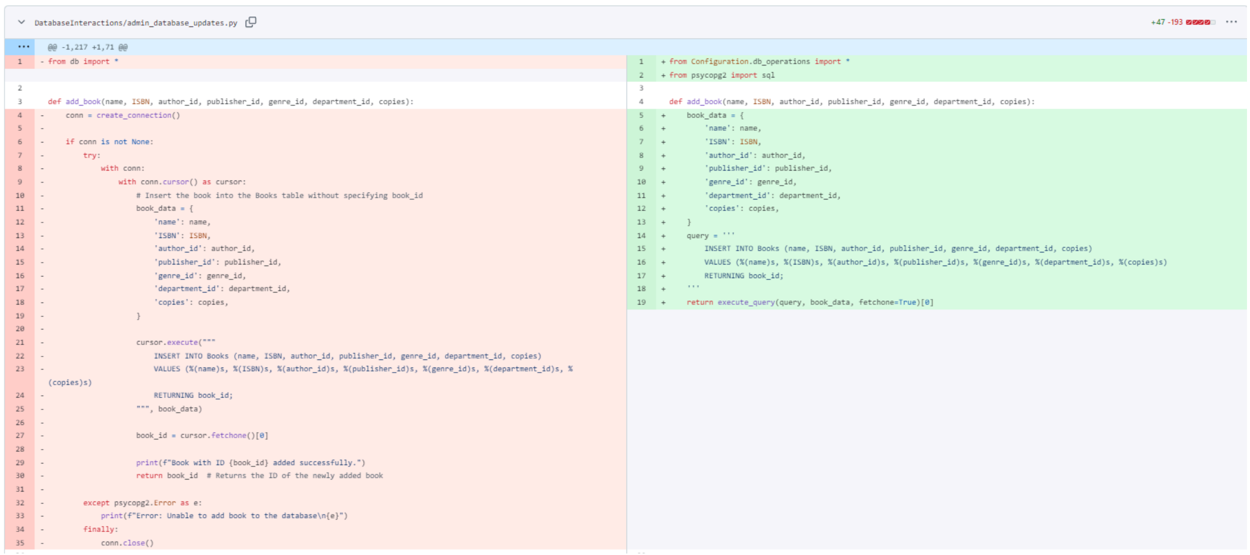
Рефакторинг повторяющихся блоков кода

Были добавлены функции execute_query и check_if_exists, которые позволили серьезно сократить повторяемость кода:



Configuration/db_operations.py +34

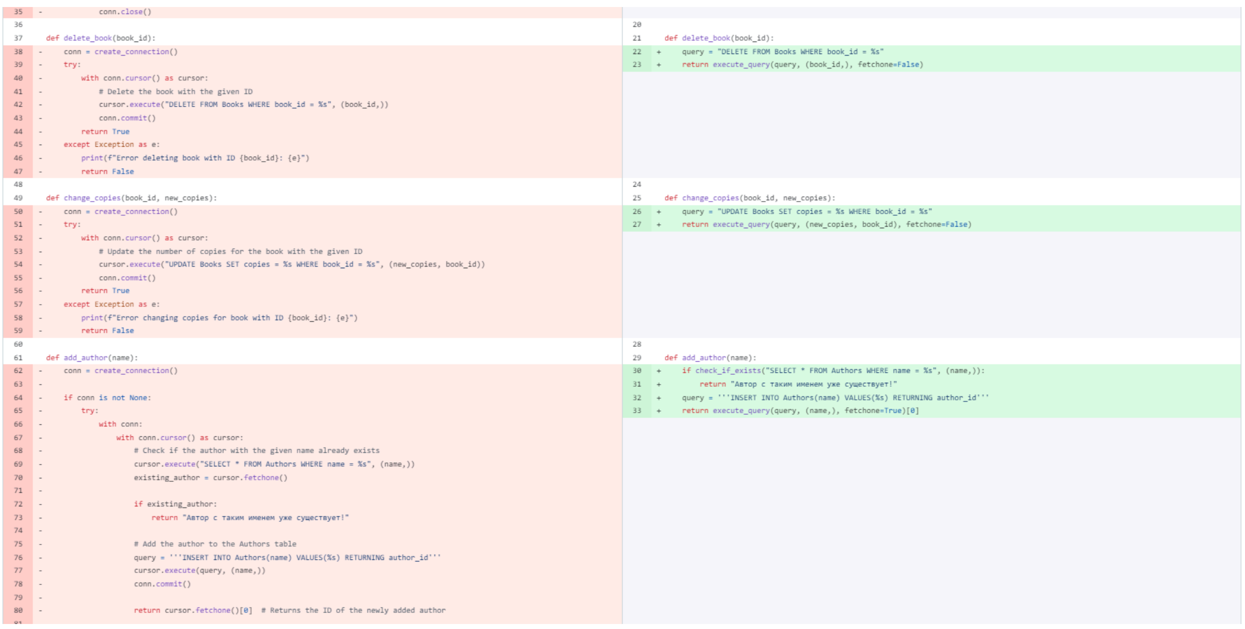
```
+ from Configuration.db import create_connection
+
+
+ def execute_query(query, params=None, fetchone=False, fetchall=False):
+     conn = create_connection()
+     result = None
+     with conn:
+         with conn.cursor() as cursor:
+             cursor.execute(query, params)
+             if fetchone:
+                 result = cursor.fetchone()
+             elif fetchall:
+                 result = cursor.fetchall()
+             conn.commit()
+     return result
+
+
+ def check_if_exists(query, params):
+     result = execute_query(query, params, fetchone=True)
+     return result is not None
+
+ def handle_db_errors(func):
+     @wraps(func)
+     def wrapper(*args, **kwargs):
+         try:
+             return func(*args, **kwargs)
+         except Exception as e:
+             logging.error(f"Error in {func.__name__}: {e}")
+             return None
+     return wrapper
```



DatabaseInteractions/admin_database_updates.py +47 -193

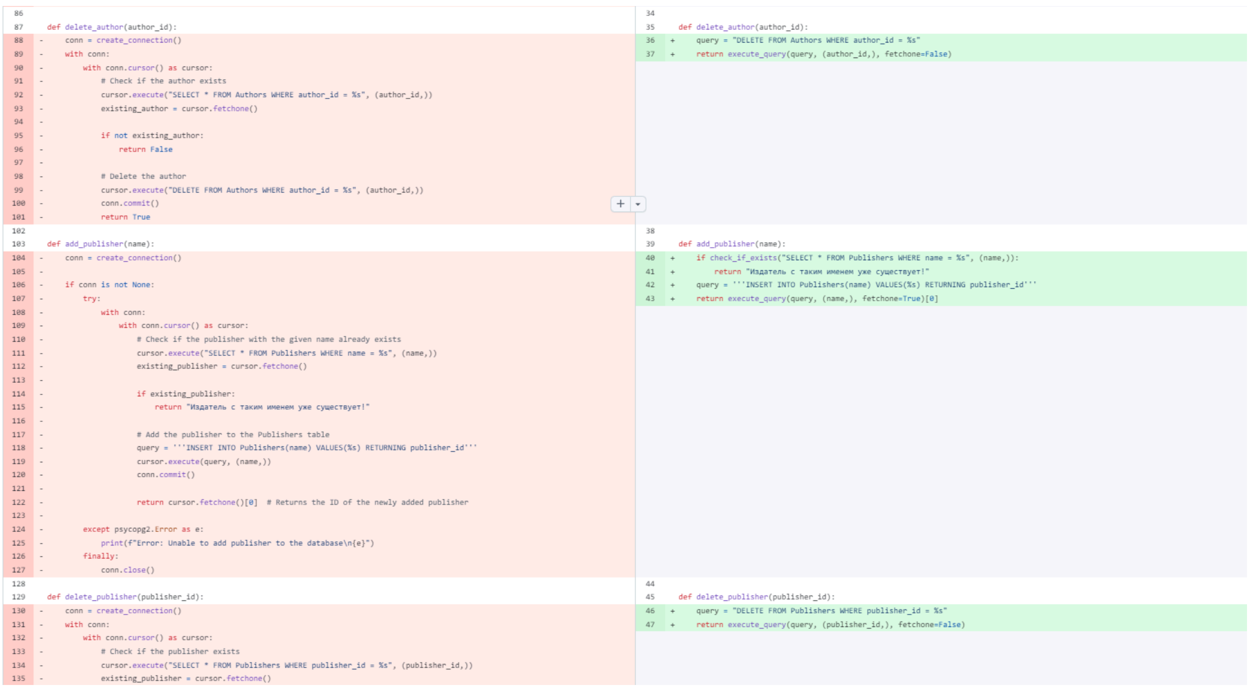
```
@@ -1,217 +1,71 @@
- from db import *

  def add_book(name, ISBN, author_id, publisher_id, genre_id, department_id, copies):
-     conn = create_connection()
-
-     if conn is not None:
-         try:
-             with conn:
-                 with conn.cursor() as cursor:
-                     # Insert the book into the Books table without specifying book_id
-                     book_data = {
-                         'name': name,
-                         'ISBN': ISBN,
-                         'author_id': author_id,
-                         'publisher_id': publisher_id,
-                         'genre_id': genre_id,
-                         'department_id': department_id,
-                         'copies': copies,
-                     }
-
-                     cursor.execute("""
-                         INSERT INTO Books (name, ISBN, author_id, publisher_id, genre_id, department_id, copies)
-                         VALUES (%(name)s, %(ISBN)s, %(author_id)s, %(publisher_id)s, %(genre_id)s, %(department_id)s, %
(copies)s)
-                         RETURNING book_id;
-                     """, book_data)
-
-                     book_id = cursor.fetchone()[0]
-
-                     print(f"Book with ID {book_id} added successfully.")
-                     return book_id  # Returns the ID of the newly added book
-
-         except psycopg2.Error as e:
-             print(f"Error: Unable to add book to the database\n{e}")
-         finally:
-             conn.close()
```

```
+ from Configuration.db_operations import *
+ from psycopg2 import sql

  def add_book(name, ISBN, author_id, publisher_id, genre_id, department_id, copies):
+     book_data = {
+         'name': name,
+         'ISBN': ISBN,
+         'author_id': author_id,
+         'publisher_id': publisher_id,
+         'genre_id': genre_id,
+         'department_id': department_id,
+         'copies': copies,
+     }
+     query = '''
+         INSERT INTO Books (name, ISBN, author_id, publisher_id, genre_id, department_id, copies)
+         VALUES (%(name)s, %(ISBN)s, %(author_id)s, %(publisher_id)s, %(genre_id)s, %(department_id)s, %(copies)s)
+         RETURNING book_id;
+     '''
+     return execute_query(query, book_data, fetchone=True)[0]
```



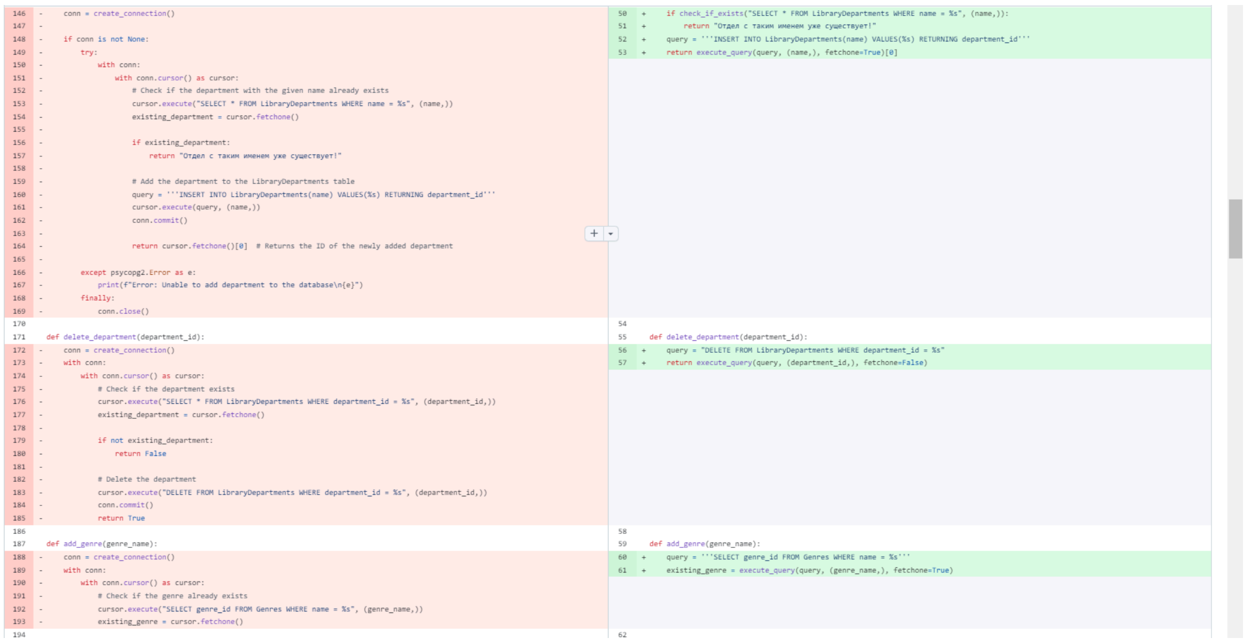
```
-             conn.close()

  def delete_book(book_id):
-     conn = create_connection()
-     try:
-         with conn.cursor() as cursor:
-             # Delete the book with the given ID
-             cursor.execute("DELETE FROM Books WHERE book_id = %s", (book_id,))
-             conn.commit()
-         return True
-     except Exception as e:
-         print(f"Error deleting book with ID {book_id}: {e}")
-         return False

  def change_copies(book_id, new_copies):
-     conn = create_connection()
-     try:
-         with conn.cursor() as cursor:
-             # Update the number of copies for the book with the given ID
-             cursor.execute("UPDATE Books SET copies = %s WHERE book_id = %s", (new_copies, book_id))
-             conn.commit()
-         return True
-     except Exception as e:
-         print(f"Error changing copies for book with ID {book_id}: {e}")
-         return False

  def add_author(name):
-     conn = create_connection()
-
-     if conn is not None:
-         try:
-             with conn:
-                 with conn.cursor() as cursor:
-                     # Check if the author with the given name already exists
-                     cursor.execute("SELECT * FROM Authors WHERE name = %s", (name,))
-                     existing_author = cursor.fetchone()
-
-                     if existing_author:
-                         return "Автор с таким именем уже существует!"
-
-                     # Add the author to the Authors table
-                     query = '''INSERT INTO Authors(name) VALUES(%s) RETURNING author_id'''
-                     cursor.execute(query, (name,))
-                     conn.commit()
-
-                     return cursor.fetchone()[0]  # Returns the ID of the newly added author
```

```

  def delete_book(book_id):
+     query = "DELETE FROM Books WHERE book_id = %s"
+     return execute_query(query, (book_id,), fetchone=False)

  def change_copies(book_id, new_copies):
+     query = "UPDATE Books SET copies = %s WHERE book_id = %s"
+     return execute_query(query, (new_copies, book_id), fetchone=False)

  def add_author(name):
+     if check_if_exists("SELECT * FROM Authors WHERE name = %s", (name,)):
+         return "Автор с таким именем уже существует!"
+     query = '''INSERT INTO Authors(name) VALUES(%s) RETURNING author_id'''
+     return execute_query(query, (name,), fetchone=True)[0]
```



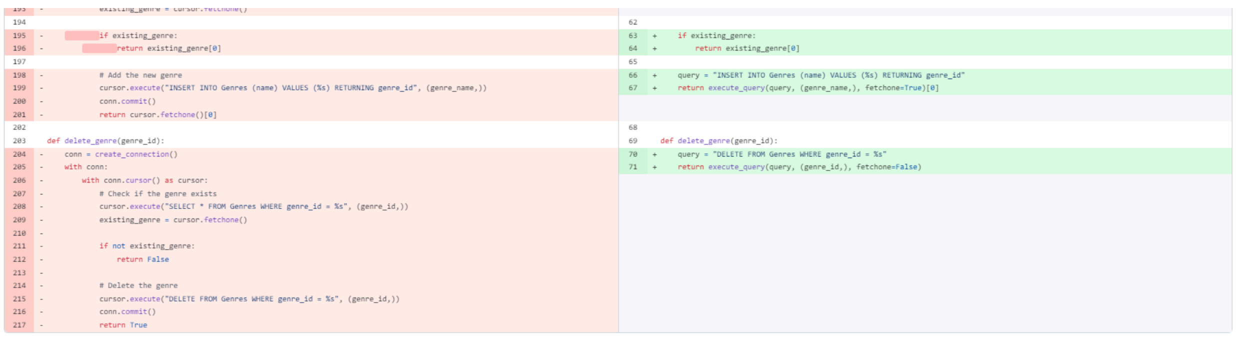
```
 def delete_author(author_id):
-    conn = create_connection()
-    with conn:
-        with conn.cursor() as cursor:
-            # Check if the author exists
-            cursor.execute("SELECT * FROM Authors WHERE author_id = %s", (author_id,))
-            existing_author = cursor.fetchone()
-
-            if not existing_author:
-                return False
-
-            # Delete the author
-            cursor.execute("DELETE FROM Authors WHERE author_id = %s", (author_id,))
-            conn.commit()
-            return True

 def add_publisher(name):
-    conn = create_connection()
-
-    if conn is not None:
-        try:
-            with conn:
-                with conn.cursor() as cursor:
-                    # Check if the publisher with the given name already exists
-                    cursor.execute("SELECT * FROM Publishers WHERE name = %s", (name,))
-                    existing_publisher = cursor.fetchone()
-
-                    if existing_publisher:
-                        return "Издатель с таким именем уже существует!"
-
-                    # Add the publisher to the Publishers table
-                    query = '''INSERT INTO Publishers(name) VALUES(%s) RETURNING publisher_id'''
-                    cursor.execute(query, (name,))
-                    conn.commit()
-
-                    return cursor.fetchone()[0]  # Returns the ID of the newly added publisher
-
-        except psycopg2.Error as e:
-            print(f"Error: Unable to add publisher to the database\n{e}")
-        finally:
-            conn.close()

 def delete_publisher(publisher_id):
-    conn = create_connection()
-    with conn:
-        with conn.cursor() as cursor:
-            # Check if the publisher exists
-            cursor.execute("SELECT * FROM Publishers WHERE publisher_id = %s", (publisher_id,))
-            existing_publisher = cursor.fetchone()
```

```
 def delete_author(author_id):
+    query = "DELETE FROM Authors WHERE author_id = %s"
+    return execute_query(query, (author_id,), fetchone=False)

 def add_publisher(name):
+    if check_if_exists("SELECT * FROM Publishers WHERE name = %s", (name,)):
+        return "Издатель с таким именем уже существует!"
+    query = '''INSERT INTO Publishers(name) VALUES(%s) RETURNING publisher_id'''
+    return execute_query(query, (name,), fetchone=True)[0]

 def delete_publisher(publisher_id):
+    query = "DELETE FROM Publishers WHERE publisher_id = %s"
+    return execute_query(query, (publisher_id,), fetchone=False)
```



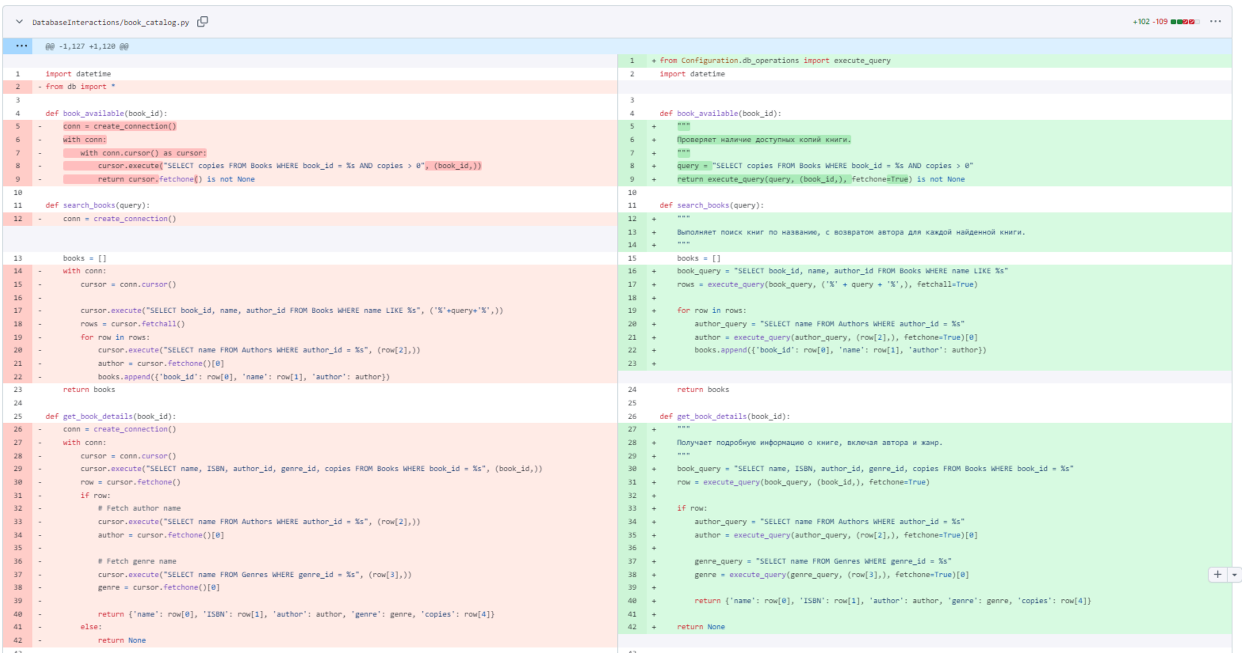
```
-    conn = create_connection()
-
-    if conn is not None:
-        try:
-            with conn:
-                with conn.cursor() as cursor:
-                    # Check if the department with the given name already exists
-                    cursor.execute("SELECT * FROM LibraryDepartments WHERE name = %s", (name,))
-                    existing_department = cursor.fetchone()
-
-                    if existing_department:
-                        return "Отдел с таким именем уже существует!"
-
-                    # Add the department to the LibraryDepartments table
-                    query = '''INSERT INTO LibraryDepartments(name) VALUES(%s) RETURNING department_id'''
-                    cursor.execute(query, (name,))
-                    conn.commit()
-
-                    return cursor.fetchone()[0]  # Returns the ID of the newly added department
-
-        except psycopg2.Error as e:
-            print(f"Error: Unable to add department to the database\n{e}")
-        finally:
-            conn.close()

 def delete_department(department_id):
-    conn = create_connection()
-    with conn:
-        with conn.cursor() as cursor:
-            # Check if the department exists
-            cursor.execute("SELECT * FROM LibraryDepartments WHERE department_id = %s", (department_id,))
-            existing_department = cursor.fetchone()
-
-            if not existing_department:
-                return False
-
-            # Delete the department
-            cursor.execute("DELETE FROM LibraryDepartments WHERE department_id = %s", (department_id,))
-            conn.commit()
-            return True

 def add_genre(genre_name):
-    conn = create_connection()
-    with conn:
-        with conn.cursor() as cursor:
-            # Check if the genre already exists
-            cursor.execute("SELECT genre_id FROM Genres WHERE name = %s", (genre_name,))
-            existing_genre = cursor.fetchone()
```

```
+    if check_if_exists("SELECT * FROM LibraryDepartments WHERE name = %s", (name,)):
+        return "Отдел с таким именем уже существует!"
+    query = '''INSERT INTO LibraryDepartments(name) VALUES(%s) RETURNING department_id'''
+    return execute_query(query, (name,), fetchone=True)[0]

 def delete_department(department_id):
+    query = "DELETE FROM LibraryDepartments WHERE department_id = %s"
+    return execute_query(query, (department_id,), fetchone=False)

 def add_genre(genre_name):
+    query = '''SELECT genre_id FROM Genres WHERE name = %s'''
+    existing_genre = execute_query(query, (genre_name,), fetchone=True)
```



```diff
-            existing_genre = cursor.fetchone()

-            if existing_genre:
-                return existing_genre[0]

-            # Add the new genre
-            cursor.execute("INSERT INTO Genres (name) VALUES (%s) RETURNING genre_id", (genre_name,))
-            conn.commit()
-            return cursor.fetchone()[0]

 def delete_genre(genre_id):
-    conn = create_connection()
-    with conn:
-        with conn.cursor() as cursor:
-            # Check if the genre exists
-            cursor.execute("SELECT * FROM Genres WHERE genre_id = %s", (genre_id,))
-            existing_genre = cursor.fetchone()
-
-            if not existing_genre:
-                return False
-
-            # Delete the genre
-            cursor.execute("DELETE FROM Genres WHERE genre_id = %s", (genre_id,))
-            conn.commit()
-            return True
```

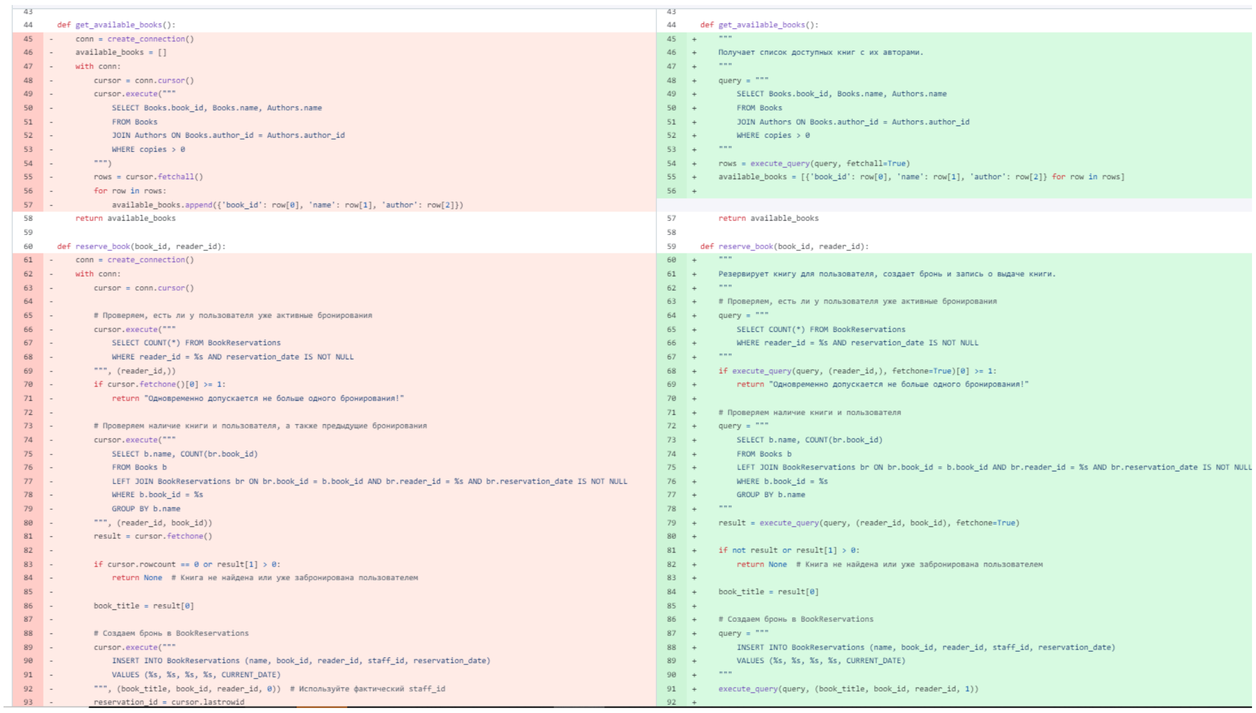
```diff

+    if existing_genre:
+        return existing_genre[0]

+    query = "INSERT INTO Genres (name) VALUES (%s) RETURNING genre_id"
+    return execute_query(query, (genre_name,), fetchone=True)[0]

 def delete_genre(genre_id):
+    query = "DELETE FROM Genres WHERE genre_id = %s"
+    return execute_query(query, (genre_id,), fetchone=False)
```



DatabaseInteractions/book_catalog.py +102 -109

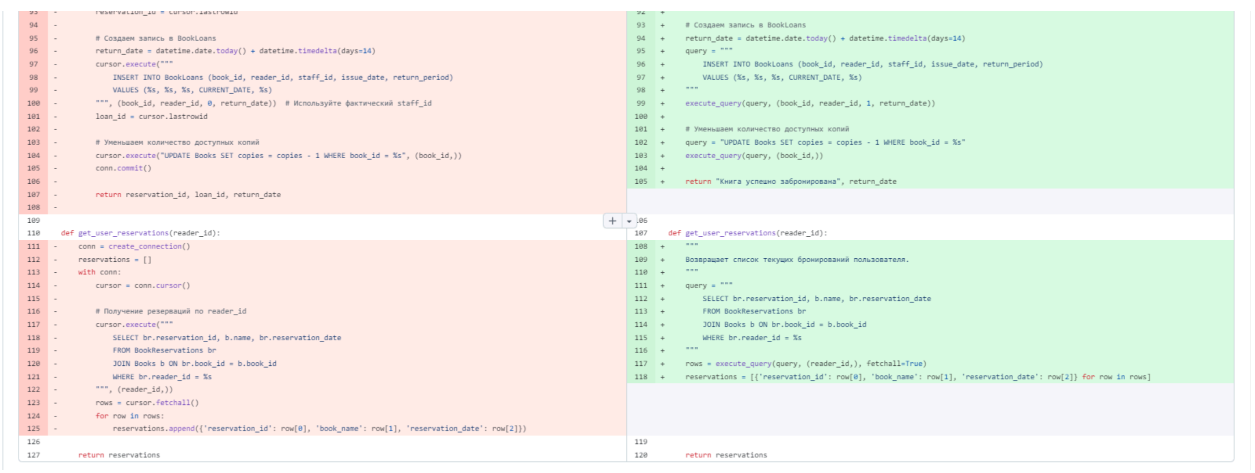
```diff
@@ -1,127 +1,120 @@
 import datetime
- from db import *

 def book_available(book_id):
-    conn = create_connection()
-    with conn:
-        with conn.cursor() as cursor:
-            cursor.execute("SELECT copies FROM Books WHERE book_id = %s AND copies > 0", (book_id,))
-            return cursor.fetchone() is not None

 def search_books(query):
-    conn = create_connection()
    books = []
-    with conn:
-        cursor = conn.cursor()
-
-        cursor.execute("SELECT book_id, name, author_id FROM Books WHERE name LIKE %s", ('%'+query+'%',))
-        rows = cursor.fetchall()
-        for row in rows:
-            cursor.execute("SELECT name FROM Authors WHERE author_id = %s", (row[2],))
-            author = cursor.fetchone()[0]
-            books.append({'book_id': row[0], 'name': row[1], 'author': author})
    return books

 def get_book_details(book_id):
-    conn = create_connection()
-    with conn:
-        cursor = conn.cursor()
-        cursor.execute("SELECT name, ISBN, author_id, genre_id, copies FROM Books WHERE book_id = %s", (book_id,))
-        row = cursor.fetchone()
-        if row:
-            # Fetch author name
-            cursor.execute("SELECT name FROM Authors WHERE author_id = %s", (row[2],))
-            author = cursor.fetchone()[0]
-
-            # Fetch genre name
-            cursor.execute("SELECT name FROM Genres WHERE genre_id = %s", (row[3],))
-            genre = cursor.fetchone()[0]
-
-            return {'name': row[0], 'ISBN': row[1], 'author': author, 'genre': genre, 'copies': row[4]}
-        else:
-            return None
```

```diff
+ from Configuration.db_operations import execute_query
 import datetime

 def book_available(book_id):
+    """
+    Проверяет наличие доступных копий книги.
+    """
+    query = "SELECT copies FROM Books WHERE book_id = %s AND copies > 0"
+    return execute_query(query, (book_id,), fetchone=True) is not None

 def search_books(query):
+    """
+    Выполняет поиск книг по названию, с возвратом автора для каждой найденной книги.
+    """
    books = []
+    book_query = "SELECT book_id, name, author_id FROM Books WHERE name LIKE %s"
+    rows = execute_query(book_query, ('%' + query + '%',), fetchall=True)
+
+    for row in rows:
+        author_query = "SELECT name FROM Authors WHERE author_id = %s"
+        author = execute_query(author_query, (row[2],), fetchone=True)[0]
+        books.append({'book_id': row[0], 'name': row[1], 'author': author})
+

    return books

 def get_book_details(book_id):
+    """
+    Получает подробную информацию о книге, включая автора и жанр.
+    """
+    book_query = "SELECT name, ISBN, author_id, genre_id, copies FROM Books WHERE book_id = %s"
+    row = execute_query(book_query, (book_id,), fetchone=True)
+
+    if row:
+        author_query = "SELECT name FROM Authors WHERE author_id = %s"
+        author = execute_query(author_query, (row[2],), fetchone=True)[0]
+
+        genre_query = "SELECT name FROM Genres WHERE genre_id = %s"
+        genre = execute_query(genre_query, (row[3],), fetchone=True)[0]
+
+        return {'name': row[0], 'ISBN': row[1], 'author': author, 'genre': genre, 'copies': row[4]}
+
+    return None
```



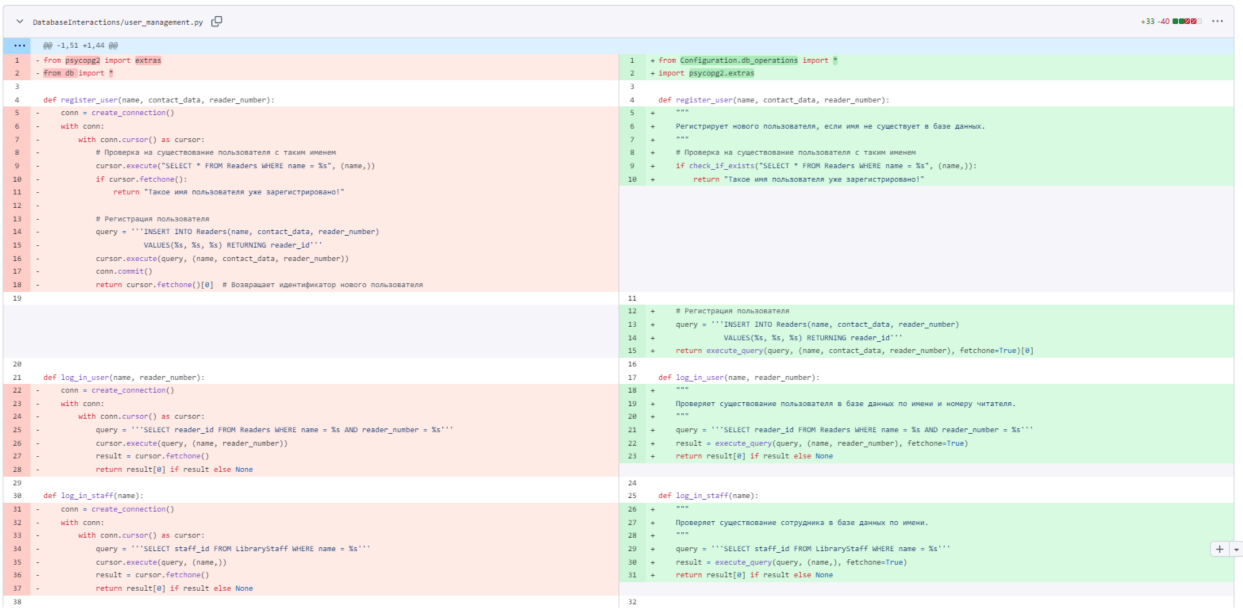
```diff
 def get_available_books():
-    conn = create_connection()
-    available_books = []
-    with conn:
-        cursor = conn.cursor()
-        cursor.execute("""
-            SELECT Books.book_id, Books.name, Authors.name
-            FROM Books
-            JOIN Authors ON Books.author_id = Authors.author_id
-            WHERE copies > 0
-        """)
-        rows = cursor.fetchall()
-        for row in rows:
-            available_books.append({'book_id': row[0], 'name': row[1], 'author': row[2]})
     return available_books

 def reserve_book(book_id, reader_id):
-    conn = create_connection()
-    with conn:
-        cursor = conn.cursor()
-
-        # Проверяем, есть ли у пользователя уже активные бронирования
-        cursor.execute("""
-            SELECT COUNT(*) FROM BookReservations
-            WHERE reader_id = %s AND reservation_date IS NOT NULL
-        """, (reader_id,))
-        if cursor.fetchone()[0] >= 1:
-            return "Одновременно допускается не больше одного бронирования!"
-
-        # Проверяем наличие книги и пользователя, а также предыдущие бронирования
-        cursor.execute("""
-            SELECT b.name, COUNT(br.book_id)
-            FROM Books b
-            LEFT JOIN BookReservations br ON br.book_id = b.book_id AND br.reader_id = %s AND br.reservation_date IS NOT NULL
-            WHERE b.book_id = %s
-            GROUP BY b.name
-        """, (reader_id, book_id))
-        result = cursor.fetchone()
-
-        if cursor.rowcount == 0 or result[1] > 0:
-            return None  # Книга не найдена или уже забронирована пользователем
-
-        book_title = result[0]
-
-        # Создаем бронь в BookReservations
-        cursor.execute("""
-            INSERT INTO BookReservations (name, book_id, reader_id, staff_id, reservation_date)
-            VALUES (%s, %s, %s, %s, CURRENT_DATE)
-        """, (book_title, book_id, reader_id, 0))  # Используйте фактический staff_id
-        reservation_id = cursor.lastrowid
```

```diff
 def get_available_books():
+    """
+    Получает список доступных книг с их авторами.
+    """
+    query = """
+        SELECT Books.book_id, Books.name, Authors.name
+        FROM Books
+        JOIN Authors ON Books.author_id = Authors.author_id
+        WHERE copies > 0
+    """
+    rows = execute_query(query, fetchall=True)
+    available_books = [{'book_id': row[0], 'name': row[1], 'author': row[2]} for row in rows]
+
     return available_books

 def reserve_book(book_id, reader_id):
+    """
+    Резервирует книгу для пользователя, создает бронь и запись о выдаче книги.
+    """
+    # Проверяем, есть ли у пользователя уже активные бронирования
+    query = """
+        SELECT COUNT(*) FROM BookReservations
+        WHERE reader_id = %s AND reservation_date IS NOT NULL
+    """
+    if execute_query(query, (reader_id,), fetchone=True)[0] >= 1:
+        return "Одновременно допускается не больше одного бронирования!"
+
+    # Проверяем наличие книги и пользователя
+    query = """
+        SELECT b.name, COUNT(br.book_id)
+        FROM Books b
+        LEFT JOIN BookReservations br ON br.book_id = b.book_id AND br.reader_id = %s AND br.reservation_date IS NOT NULL
+        WHERE b.book_id = %s
+        GROUP BY b.name
+    """
+    result = execute_query(query, (reader_id, book_id), fetchone=True)
+
+    if not result or result[1] > 0:
+        return None  # Книга не найдена или уже забронирована пользователем
+
+    book_title = result[0]
+
+    # Создаем бронь в BookReservations
+    query = """
+        INSERT INTO BookReservations (name, book_id, reader_id, staff_id, reservation_date)
+        VALUES (%s, %s, %s, %s, CURRENT_DATE)
+    """
+    execute_query(query, (book_title, book_id, reader_id, 1))
+
```

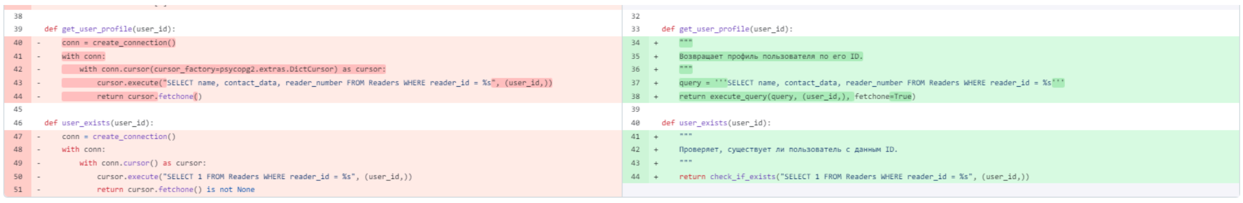


```diff
-        reservation_id = cursor.lastrowid
-
-        # Создаем запись в BookLoans
-        return_date = datetime.date.today() + datetime.timedelta(days=14)
-        cursor.execute("""
-            INSERT INTO BookLoans (book_id, reader_id, staff_id, issue_date, return_period)
-            VALUES (%s, %s, %s, CURRENT_DATE, %s)
-        """, (book_id, reader_id, 0, return_date))  # Используйте фактический staff_id
-        loan_id = cursor.lastrowid
-
-        # Уменьшаем количество доступных копий
-        cursor.execute("UPDATE Books SET copies = copies - 1 WHERE book_id = %s", (book_id,))
-        conn.commit()
-
-        return reservation_id, loan_id, return_date
-

 def get_user_reservations(reader_id):
-    conn = create_connection()
-    reservations = []
-    with conn:
-        cursor = conn.cursor()
-
-        # Получение резерваций по reader_id
-        cursor.execute("""
-            SELECT br.reservation_id, b.name, br.reservation_date
-            FROM BookReservations br
-            JOIN Books b ON br.book_id = b.book_id
-            WHERE br.reader_id = %s
-        """, (reader_id,))
-        rows = cursor.fetchall()
-        for row in rows:
-            reservations.append({'reservation_id': row[0], 'book_name': row[1], 'reservation_date': row[2]})

     return reservations
```

```diff
+
+    # Создаем запись в BookLoans
+    return_date = datetime.date.today() + datetime.timedelta(days=14)
+    query = """
+        INSERT INTO BookLoans (book_id, reader_id, staff_id, issue_date, return_period)
+        VALUES (%s, %s, %s, CURRENT_DATE, %s)
+    """
+    execute_query(query, (book_id, reader_id, 1, return_date))
+
+    # Уменьшаем количество доступных копий
+    query = "UPDATE Books SET copies = copies - 1 WHERE book_id = %s"
+    execute_query(query, (book_id,))
+
+    return "Книга успешно забронирована", return_date

 def get_user_reservations(reader_id):
+    """
+    Возвращает список текущих бронирований пользователя.
+    """
+    query = """
+        SELECT br.reservation_id, b.name, br.reservation_date
+        FROM BookReservations br
+        JOIN Books b ON br.book_id = b.book_id
+        WHERE br.reader_id = %s
+    """
+    rows = execute_query(query, (reader_id,), fetchall=True)
+    reservations = [{'reservation_id': row[0], 'book_name': row[1], 'reservation_date': row[2]} for row in rows]

     return reservations
```

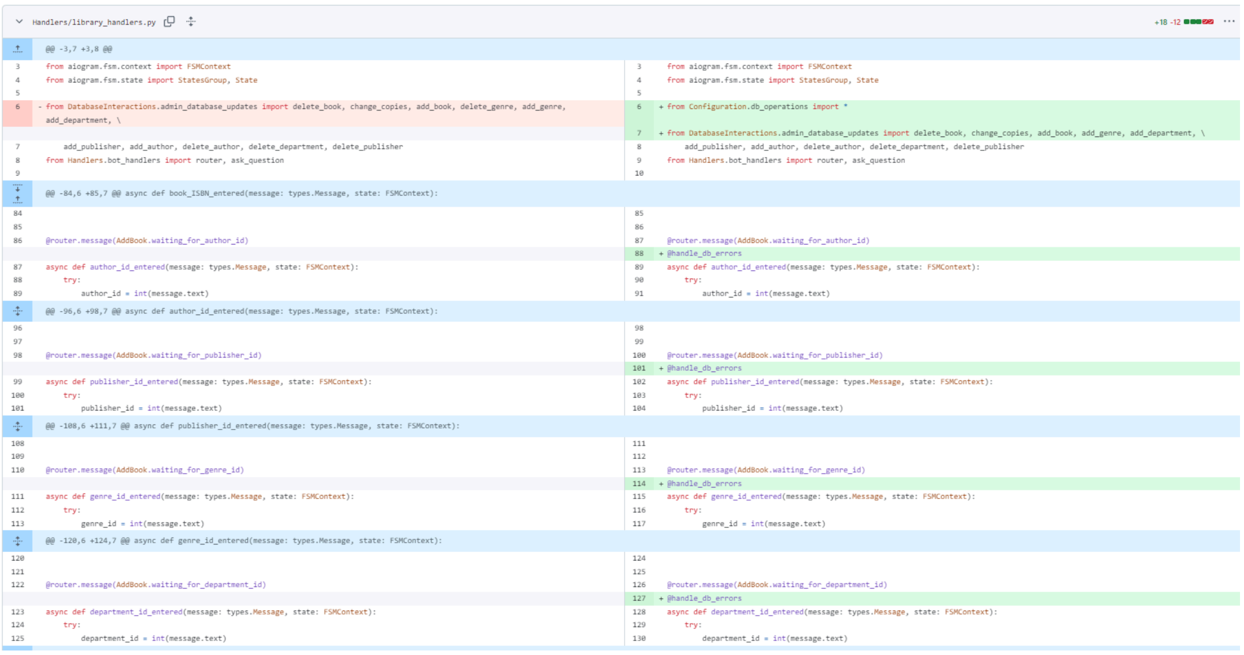


```diff
DatabaseInteractions/user_management.py                                   +33 -40

@@ -1,51 +1,44 @@
- from psycopg2 import extras
- from db import *
+ from Configuration.db_operations import *
+ import psycopg2.extras

  def register_user(name, contact_data, reader_number):
-     conn = create_connection()
-     with conn:
-         with conn.cursor() as cursor:
-             # Проверка на существование пользователя с таким именем
-             cursor.execute("SELECT * FROM Readers WHERE name = %s", (name,))
-             if cursor.fetchone():
-                 return "Такое имя пользователя уже зарегистрировано!"
-
-             # Регистрация пользователя
-             query = '''INSERT INTO Readers(name, contact_data, reader_number)
-                        VALUES(%s, %s, %s) RETURNING reader_id'''
-             cursor.execute(query, (name, contact_data, reader_number))
-             conn.commit()
-             return cursor.fetchone()[0]  # Возвращает идентификатор нового пользователя
+     """
+     Регистрирует нового пользователя, если имя не существует в базе данных.
+     """
+     # Проверка на существование пользователя с таким именем
+     if check_if_exists("SELECT * FROM Readers WHERE name = %s", (name,)):
+         return "Такое имя пользователя уже зарегистрировано!"

+     # Регистрация пользователя
+     query = '''INSERT INTO Readers(name, contact_data, reader_number)
+                VALUES(%s, %s, %s) RETURNING reader_id'''
+     return execute_query(query, (name, contact_data, reader_number), fetchone=True)[0]

  def log_in_user(name, reader_number):
-     conn = create_connection()
-     with conn:
-         with conn.cursor() as cursor:
-             query = '''SELECT reader_id FROM Readers WHERE name = %s AND reader_number = %s'''
-             cursor.execute(query, (name, reader_number))
-             result = cursor.fetchone()
-             return result[0] if result else None
+     """
+     Проверяет существование пользователя в базе данных по имени и номеру читателя.
+     """
+     query = '''SELECT reader_id FROM Readers WHERE name = %s AND reader_number = %s'''
+     result = execute_query(query, (name, reader_number), fetchone=True)
+     return result[0] if result else None

  def log_in_staff(name):
-     conn = create_connection()
-     with conn:
-         with conn.cursor() as cursor:
-             query = '''SELECT staff_id FROM LibraryStaff WHERE name = %s'''
-             cursor.execute(query, (name,))
-             result = cursor.fetchone()
-             return result[0] if result else None
+     """
+     Проверяет существование сотрудника в базе данных по имени.
+     """
+     query = '''SELECT staff_id FROM LibraryStaff WHERE name = %s'''
+     result = execute_query(query, (name,), fetchone=True)
+     return result[0] if result else None

```

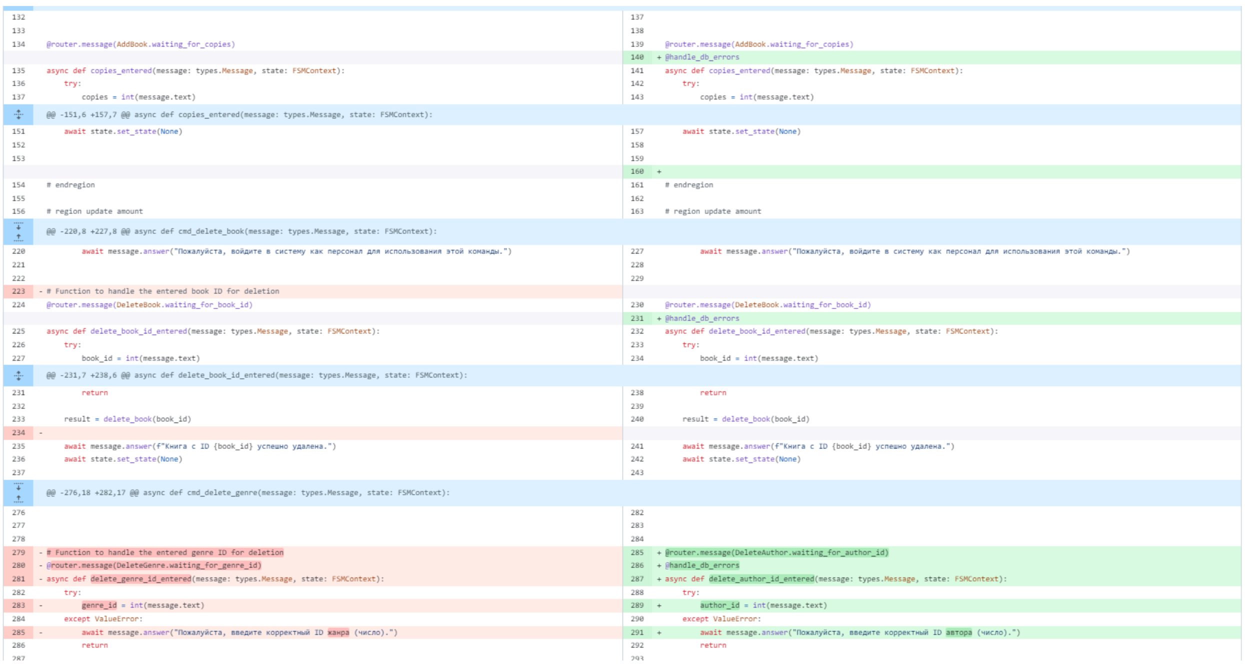


```diff

  def get_user_profile(user_id):
-     conn = create_connection()
-     with conn:
-         with conn.cursor(cursor_factory=psycopg2.extras.DictCursor) as cursor:
-             cursor.execute("SELECT name, contact_data, reader_number FROM Readers WHERE reader_id = %s", (user_id,))
-             return cursor.fetchone()
+     """
+     Возвращает профиль пользователя по его ID.
+     """
+     query = '''SELECT name, contact_data, reader_number FROM Readers WHERE reader_id = %s'''
+     return execute_query(query, (user_id,), fetchone=True)

  def user_exists(user_id):
-     conn = create_connection()
-     with conn:
-         with conn.cursor() as cursor:
-             cursor.execute("SELECT 1 FROM Readers WHERE reader_id = %s", (user_id,))
-             return cursor.fetchone() is not None
+     """
+     Проверяет, существует ли пользователь с данным ID.
+     """
+     return check_if_exists("SELECT 1 FROM Readers WHERE reader_id = %s", (user_id,))
```

Также был добавлен декоратор @handle_db_errors для обработки и логирования общих ошибок:

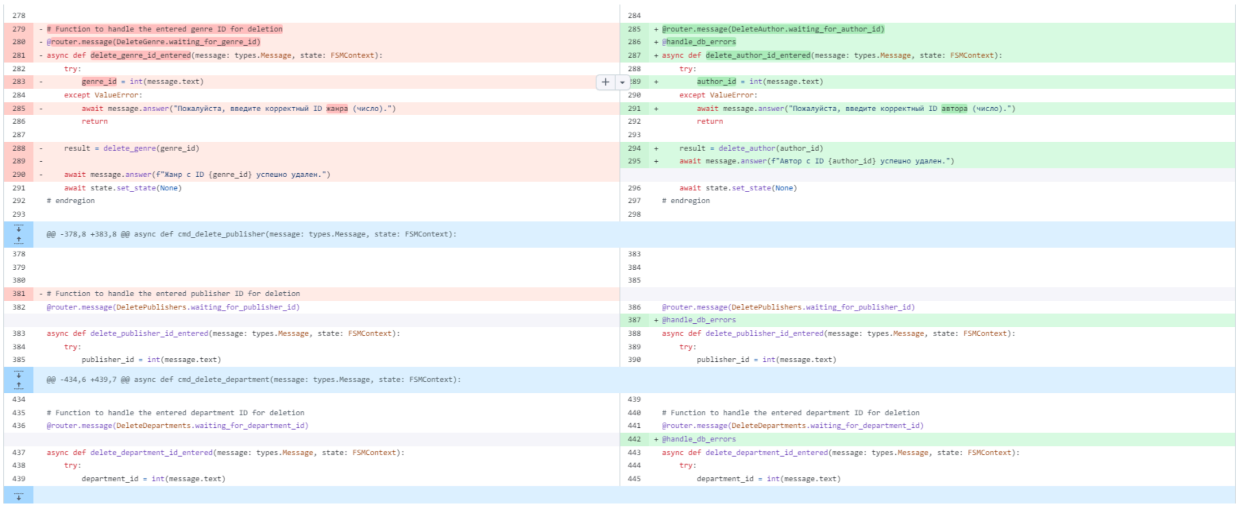


```diff
Handlers/library_handlers.py                                              +18 -12

@@ -3,7 +3,8 @@
  from aiogram.fsm.context import FSMContext
  from aiogram.fsm.state import StatesGroup, State

- from DatabaseInteractions.admin_database_updates import delete_book, change_copies, add_book, delete_genre, add_genre,
  add_department, \
+ from Configuration.db_operations import *

+ from DatabaseInteractions.admin_database_updates import delete_book, change_copies, add_book, add_genre, add_department, \
      add_publisher, add_author, delete_author, delete_department, delete_publisher
  from Handlers.bot_handlers import router, ask_question

@@ -84,6 +85,7 @@ async def book_ISBN_entered(message: types.Message, state: FSMContext):


  @router.message(AddBook.waiting_for_author_id)
+ @handle_db_errors
  async def author_id_entered(message: types.Message, state: FSMContext):
      try:
          author_id = int(message.text)
@@ -96,6 +98,7 @@ async def author_id_entered(message: types.Message, state: FSMContext):


  @router.message(AddBook.waiting_for_publisher_id)
+ @handle_db_errors
  async def publisher_id_entered(message: types.Message, state: FSMContext):
      try:
          publisher_id = int(message.text)
@@ -108,6 +111,7 @@ async def publisher_id_entered(message: types.Message, state: FSMContext):


  @router.message(AddBook.waiting_for_genre_id)
+ @handle_db_errors
  async def genre_id_entered(message: types.Message, state: FSMContext):
      try:
          genre_id = int(message.text)
@@ -120,6 +124,7 @@ async def genre_id_entered(message: types.Message, state: FSMContext):


  @router.message(AddBook.waiting_for_department_id)
+ @handle_db_errors
  async def department_id_entered(message: types.Message, state: FSMContext):
      try:
          department_id = int(message.text)
```



```diff


@router.message(AddBook.waiting_for_copies)
async def copies_entered(message: types.Message, state: FSMContext):
    try:
        copies = int(message.text)
@@ -151,6 +157,7 @@ async def copies_entered(message: types.Message, state: FSMContext):
    await state.set_state(None)


# endregion

# region update amount
@@ -220,8 +227,8 @@ async def cmd_delete_book(message: types.Message, state: FSMContext):
        await message.answer("Пожалуйста, войдите в систему как персонал для использования этой команды.")


- # Function to handle the entered book ID for deletion
@router.message(DeleteBook.waiting_for_book_id)
async def delete_book_id_entered(message: types.Message, state: FSMContext):
    try:
        book_id = int(message.text)
@@ -231,7 +238,6 @@ async def delete_book_id_entered(message: types.Message, state: FSMContext):
        return

    result = delete_book(book_id)
-
    await message.answer(f"Книга с ID {book_id} успешно удалена.")
    await state.set_state(None)

@@ -276,18 +282,17 @@ async def cmd_delete_genre(message: types.Message, state: FSMContext):



- # Function to handle the entered genre ID for deletion
- @router.message(DeleteGenre.waiting_for_genre_id)
- async def delete_genre_id_entered(message: types.Message, state: FSMContext):
    try:
-           genre_id = int(message.text)
    except ValueError:
-           await message.answer("Пожалуйста, введите корректный ID жанра (число).")
        return
```

```diff


@router.message(AddBook.waiting_for_copies)
+ @handle_db_errors
async def copies_entered(message: types.Message, state: FSMContext):
    try:
        copies = int(message.text)

    await state.set_state(None)


+
# endregion

# region update amount

        await message.answer("Пожалуйста, войдите в систему как персонал для использования этой команды.")


@router.message(DeleteBook.waiting_for_book_id)
+ @handle_db_errors
async def delete_book_id_entered(message: types.Message, state: FSMContext):
    try:
        book_id = int(message.text)

        return

    result = delete_book(book_id)
    await message.answer(f"Книга с ID {book_id} успешно удалена.")
    await state.set_state(None)





+ @router.message(DeleteAuthor.waiting_for_author_id)
+ @handle_db_errors
+ async def delete_author_id_entered(message: types.Message, state: FSMContext):
    try:
+           author_id = int(message.text)
    except ValueError:
+           await message.answer("Пожалуйста, введите корректный ID автора (число).")
        return
```



```diff

- # Function to handle the entered genre ID for deletion
- @router.message(DeleteGenre.waiting_for_genre_id)
- async def delete_genre_id_entered(message: types.Message, state: FSMContext):
    try:
-           genre_id = int(message.text)
    except ValueError:
-           await message.answer("Пожалуйста, введите корректный ID жанра (число).")
        return

-   result = delete_genre(genre_id)
-
-   await message.answer(f"Жанр с ID {genre_id} успешно удален.")
    await state.set_state(None)
# endregion

@@ -378,8 +383,8 @@ async def cmd_delete_publisher(message: types.Message, state: FSMContext):



- # Function to handle the entered publisher ID for deletion
@router.message(DeletePublishers.waiting_for_publisher_id)
async def delete_publisher_id_entered(message: types.Message, state: FSMContext):
    try:
        publisher_id = int(message.text)
@@ -434,6 +439,7 @@ async def cmd_delete_department(message: types.Message, state: FSMContext):

# Function to handle the entered department ID for deletion
@router.message(DeleteDepartments.waiting_for_department_id)
async def delete_department_id_entered(message: types.Message, state: FSMContext):
    try:
        department_id = int(message.text)
```

```diff

+ @router.message(DeleteAuthor.waiting_for_author_id)
+ @handle_db_errors
+ async def delete_author_id_entered(message: types.Message, state: FSMContext):
    try:
+           author_id = int(message.text)
    except ValueError:
+           await message.answer("Пожалуйста, введите корректный ID автора (число).")
        return

+   result = delete_author(author_id)
+   await message.answer(f"Автор с ID {author_id} успешно удален.")
    await state.set_state(None)
# endregion




@router.message(DeletePublishers.waiting_for_publisher_id)
+ @handle_db_errors
async def delete_publisher_id_entered(message: types.Message, state: FSMContext):
    try:
        publisher_id = int(message.text)


# Function to handle the entered department ID for deletion
@router.message(DeleteDepartments.waiting_for_department_id)
+ @handle_db_errors
async def delete_department_id_entered(message: types.Message, state: FSMContext):
    try:
        department_id = int(message.text)
```