

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.

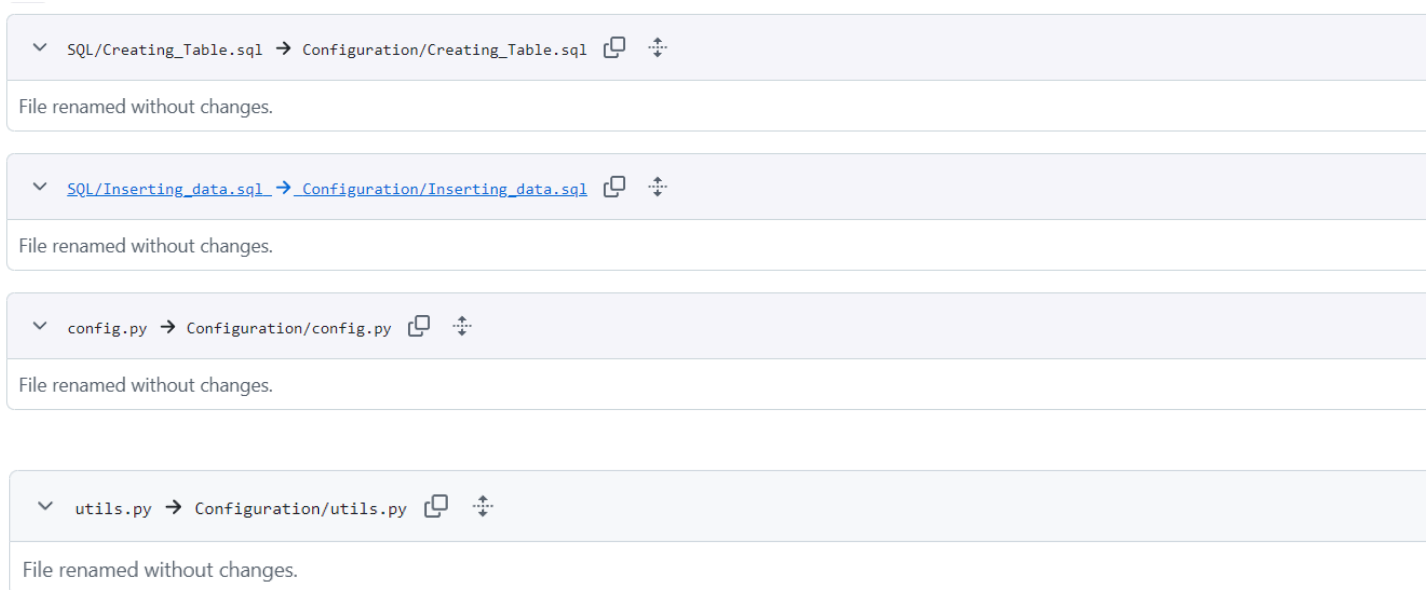
Санкт-Петербург, 2024 г.

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



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

```

1  - from db import *

2

3  def add_book(name, isbn, author_id, publisher_id, genre_id, department_id, copies):
4      - conn = create_connection()
5      - if conn is not None:
6          - try:
7              with conn.cursor() as cursor:
8                  # Insert the book into the Books table without specifying book_id
9                  book_data = {
10                     'name': name,
11                     'isbn': isbn,
12                     'author_id': author_id,
13                     'publisher_id': publisher_id,
14                     'genre_id': genre_id,
15                     'department_id': department_id,
16                     'copies': copies,
17                 }
18             cursor.execute("""
19                 INSERT INTO Books (name, isbn, author_id, publisher_id, genre_id, department_id, copies)
20                 VALUES (%s,%s,%s,%s,%s,%s,%s)
21             """, book_data)
22             book_id = cursor.fetchone()[0]
23             print(f"Book with ID {book_id} added successfully.")
24             return book_id # Returns the ID of the newly added book
25         except psycopg2.Error as e:
26             print(f"Error: Unable to add book to the database({e})")
27         finally:
28             conn.close()
29
30
31
32
33
34
35
36
37 def delete_book(book_id):
38     - conn = create_connection()
39     - try:
40         with conn.cursor() as cursor:
41             # Delete the book with the given ID
42             cursor.execute("DELETE FROM Books WHERE book_id = %s", (book_id,))
43             conn.commit()
44             return True
45     except Exception as e:
46         print(f"Error deleting book with ID {book_id}: {e}")
47     return False
48
49 def change_copies(book_id, new_copies):
50     - conn = create_connection()
51     - try:
52         with conn.cursor() as cursor:
53             # Update the number of copies for the book with the given ID
54             cursor.execute("UPDATE Books SET copies = %s WHERE book_id = %s", (new_copies, book_id))
55             conn.commit()
56             return True
57     except Exception as e:
58         print(f"Error changing copies for book with ID {book_id}: {e}")
59     return False
60
61 def add_author(name):
62     - conn = create_connection()
63     - if conn is not None:
64         - try:
65             with conn.cursor() as cursor:
66                 # Check if the author with the given name already exists
67                 cursor.execute("SELECT * FROM Authors WHERE name = %s", (name,))
68                 existing_author = cursor.fetchone()
69                 if existing_author:
70                     return "Автор с таким именем уже существует!"
71                 # Add the author to the Authors table
72                 query = "INSERT INTO Authors(name) VALUES(%s) RETURNING author_id"
73                 cursor.execute(query, (name,))
74                 conn.commit()
75                 return cursor.fetchone()[0] # Returns the ID of the newly added author
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

DatabaseInteractions/admin_database_updates.py
+47

```

1  - from Configuration.db_operations import *
2  - from psycopg2 import sql

3  def add_book(name, isbn, author_id, publisher_id, genre_id, department_id, copies):
4      book_data = {
5          'name': name,
6          'isbn': isbn,
7          'author_id': author_id,
8          'publisher_id': publisher_id,
9          'genre_id': genre_id,
10         'department_id': department_id,
11         'copies': copies,
12     }
13     query = """
14         INSERT INTO Books (name, isbn, author_id, publisher_id, genre_id, department_id, copies)
15         VALUES (%s,%s,%s,%s,%s,%s,%s)
16         RETURNING book_id;
17     """
18     return execute_query(query, book_data, fetchone=True)[0]
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

<pre> 86 87 def delete_author(author_id): 88 conn = create_connection() 89 with conn: 90 with conn.cursor() as cursor: 91 # Check if the author exists 92 cursor.execute("SELECT * FROM Authors WHERE author_id = %s", (author_id,)) 93 existing_author = cursor.fetchone() 94 95 if not existing_author: 96 return False 97 98 # Delete the author 99 cursor.execute("DELETE FROM Authors WHERE author_id = %s", (author_id,)) 100 conn.commit() 101 return True </pre>	<pre> 34 def delete_author(author_id): 35 query = "DELETE FROM Authors WHERE author_id = %s" 36 + return execute_query(query, (author_id,), fetchone=False) </pre>
<pre> 102 103 def add_publisher(name): 104 conn = create_connection() 105 106 if conn is not None: 107 try: 108 with conn: 109 with conn.cursor() as cursor: 110 # Check if the publisher with the given name already exists 111 cursor.execute("SELECT * FROM Publishers WHERE name = %s", (name,)) 112 existing_publisher = cursor.fetchone() 113 114 if existing_publisher: 115 return "Издатель с таким именем уже существует!" 116 117 # Add the publisher to the Publishers table 118 query = '''INSERT INTO Publishers(name) VALUES(%s) RETURNING publisher_id''' 119 cursor.execute(query, (name,)) 120 conn.commit() 121 122 return cursor.fetchone()[0] # Returns the ID of the newly added publisher 123 124 except psycopg2.Error as e: 125 print(f"Error: Unable to add publisher to the database\n(e)") 126 finally: 127 conn.close() </pre>	<pre> 38 39 def add_publisher(name): 40 + if check_if_exists("SELECT * FROM Publishers WHERE name = %s", (name,)): 41 + return "Издатель с таким именем уже существует!" 42 + query = '''INSERT INTO Publishers(name) VALUES(%s) RETURNING publisher_id''' 43 + return execute_query(query, (name,), fetchone=True)[0] </pre>
<pre> 128 129 def delete_publisher(publisher_id): 130 conn = create_connection() 131 with conn: 132 with conn.cursor() as cursor: 133 # Check if the publisher exists 134 cursor.execute("SELECT * FROM Publishers WHERE publisher_id = %s", (publisher_id,)) 135 existing_publisher = cursor.fetchone() </pre>	<pre> 44 45 def delete_publisher(publisher_id): 46 + query = "DELETE FROM Publishers WHERE publisher_id = %s" 47 + return execute_query(query, (publisher_id,), fetchone=False) </pre>
<pre> 146 conn = create_connection() 147 148 if conn is not None: 149 try: 150 with conn: 151 with conn.cursor() as cursor: 152 # Check if the department with the given name already exists 153 cursor.execute("SELECT * FROM LibraryDepartments WHERE name = %s", (name,)) 154 existing_department = cursor.fetchone() 155 156 if existing_department: 157 return "Отдел с таким именем уже существует!" 158 159 # Add the department to the LibraryDepartments table 160 query = '''INSERT INTO LibraryDepartments(name) VALUES(%s) RETURNING department_id''' 161 cursor.execute(query, (name,)) 162 conn.commit() 163 164 return cursor.fetchone()[0] # Returns the ID of the newly added department 165 166 except psycopg2.Error as e: 167 print(f"Error: Unable to add department to the database\n(e)") 168 finally: 169 conn.close() </pre>	<pre> 50 + if check_if_exists("SELECT * FROM LibraryDepartments WHERE name = %s", (name,)): 51 + return "Отдел с таким именем уже существует!" 52 + query = '''INSERT INTO LibraryDepartments(name) VALUES(%s) RETURNING department_id''' 53 + return execute_query(query, (name,), fetchone=True)[0] </pre>
<pre> 171 172 def delete_department(department_id): 173 conn = create_connection() 174 with conn: 175 with conn.cursor() as cursor: 176 # Check if the department exists 177 cursor.execute("SELECT * FROM LibraryDepartments WHERE department_id = %s", (department_id,)) 178 existing_department = cursor.fetchone() 179 180 if not existing_department: 181 return False 182 183 # Delete the department 184 cursor.execute("DELETE FROM LibraryDepartments WHERE department_id = %s", (department_id,)) 185 conn.commit() 186 return True </pre>	<pre> 54 55 def delete_department(department_id): 56 + query = "DELETE FROM LibraryDepartments WHERE department_id = %s" 57 + return execute_query(query, (department_id,), fetchone=False) </pre>
<pre> 187 188 def add_genre(genre_name): 189 conn = create_connection() 190 with conn: 191 with conn.cursor() as cursor: 192 # Check if the genre already exists 193 cursor.execute("SELECT genre_id FROM Genres WHERE name = %s", (genre_name,)) 194 existing_genre = cursor.fetchone() </pre>	<pre> 58 59 def add_genre(genre_name): 60 + query = '''SELECT genre_id FROM Genres WHERE name = %s''' 61 + existing_genre = execute_query(query, (genre_name,), fetchone=True) </pre>

193 -	existing_genre = cursor.fetchone()	62	
194 -		63 +	if existing_genre:
195 -	if existing_genre:	64 +	return existing_genre[0]
196 -	return existing_genre[0]	65	
197 -		66 +	query = "INSERT INTO Genres (name) VALUES (%s) RETURNING genre_id"
198 -	# Add the new genre	67 +	return execute_query(query, (genre_name,), fetchone=True)[0]
199 -	cursor.execute("INSERT INTO Genres (name) VALUES (%s) RETURNING genre_id", (genre_name,))		
200 -	conn.commit()		
201 -	return cursor.fetchone()[0]		
202 -		68	
203	def delete_genre(genre_id):	69	def delete_genre(genre_id):
204 -	conn = create_connection()	70 +	query = "DELETE FROM Genres WHERE genre_id = %s"
205 -	with conn:	71 +	return execute_query(query, (genre_id,), fetchone=False)
206 -	with conn.cursor() as cursor:		
207 -	# Check if the genre exists		
208 -	cursor.execute("SELECT * FROM Genres WHERE genre_id = %s", (genre_id,))		
209 -	existing_genre = cursor.fetchone()		
210 -			
211 -	if not existing_genre:		
212 -	return False		
213 -			
214 -	# Delete the genre		
215 -	cursor.execute("DELETE FROM Genres WHERE genre_id = %s", (genre_id,))		
216 -	conn.commit()		
217 -	return True		

DatabaseInteractions/book_catalog.py		+102 -109	
@@ -1,127 +1,120 @@			
1	import datetime	1	from Configuration.db_operations import execute_query
2	from db import *	2	import datetime
3		3	
4	def book_available(book_id):	4	def book_available(book_id):
5	conn = create_connection()	5	"""
6	with conn:	6	Проверяет наличие доступных копий книги.
7	with conn.cursor() as cursor:	7	"""
8	cursor.execute("SELECT copies FROM Books WHERE book_id = %s AND copies > 0", (book_id,))	8	query = "SELECT copies FROM Books WHERE book_id = %s AND copies > 0"
9	return cursor.fetchone() is not None	9	return execute_query(query, (book_id,), fetchone=True) is not None
10		10	
11	def search_books(query):	11	def search_books(query):
12	conn = create_connection()	12	"""
13		13	Возвращает поиск книг по названию, с возвратом автора для каждой найденной книги.
14	books = []	14	"""
15	with conn:	15	books = []
16	cursor = conn.cursor()	16	book_query = "SELECT book_id, name, author_id FROM Books WHERE name LIKE %s"
17	cursor.execute("SELECT book_id, name, author_id FROM Books WHERE name LIKE %s", ('%' + query + '%',))	17	rows = execute_query(book_query, ('%' + query + '%',), fetchall=True)
18	rows = cursor.fetchall()	18	
19	for row in rows:	19	for row in rows:
20	cursor.execute("SELECT name FROM Authors WHERE author_id = %s", (row[2],))	20	author_query = "SELECT name FROM Authors WHERE author_id = %s"
21	author = cursor.fetchone()[0]	21	author = execute_query(author_query, (row[2],), fetchone=True)[0]
22	books.append({'book_id': row[0], 'name': row[1], 'author': author})	22	books.append({'book_id': row[0], 'name': row[1], 'author': author})
23		23	
24	return books	24	return books
25	def get_book_details(book_id):	25	def get_book_details(book_id):
26	conn = create_connection()	26	"""
27	with conn:	27	Получает подробные данные о книге, включая автора и жанр.
28	cursor = conn.cursor()	28	"""
29	cursor.execute("SELECT name, ISBN, author_id, genre_id, copies FROM Books WHERE book_id = %s", (book_id,))	29	book_query = "SELECT name, ISBN, author_id, genre_id, copies FROM Books WHERE book_id = %s"
30	row = cursor.fetchone()	30	row = execute_query(book_query, (book_id,), fetchone=True)
31	if row:	31	
32	# Fetch author name	32	
33	cursor.execute("SELECT name FROM Authors WHERE author_id = %s", (row[2],))	33	if row:
34	author = cursor.fetchone()[0]	34	author_query = "SELECT name FROM Authors WHERE author_id = %s"
35		35	author = execute_query(author_query, (row[2],), fetchone=True)[0]
36	# Fetch genre name	36	
37	cursor.execute("SELECT name FROM Genres WHERE genre_id = %s", (row[3],))	37	genre_query = "SELECT name FROM Genres WHERE genre_id = %s"
38	genre = cursor.fetchone()[0]	38	genre = execute_query(genre_query, (row[3],), fetchone=True)[0]
39		39	
40	return {'name': row[0], 'ISBN': row[1], 'author': author, 'genre': genre, 'copies': row[4]}	40	return {'name': row[0], 'ISBN': row[1], 'author': author, 'genre': genre, 'copies': row[4]}
41	else:	41	
42	return None	42	return None

<pre> 43 44 def get_available_books(): 45 conn = create_connection() 46 available_books = [] 47 with conn: 48 cursor = conn.cursor() 49 cursor.execute(""" 50 SELECT Books.book_id, Books.name, Authors.name 51 FROM Books 52 JOIN Authors ON Books.author_id = Authors.author_id 53 WHERE copies > 0 54 """) 55 rows = cursor.fetchall() 56 for row in rows: 57 available_books.append({'book_id': row[0], 'name': row[1], 'author': row[2]}) 58 return available_books 59 60 def reserve_book(book_id, reader_id): 61 conn = create_connection() 62 with conn: 63 cursor = conn.cursor() 64 65 # Проверим, есть ли у пользователя уже активные бронирования 66 cursor.execute(""" 67 SELECT COUNT(*) FROM BookReservations 68 WHERE reader_id = %s AND reservation_date IS NOT NULL 69 """, (reader_id,)) 70 if cursor.fetchone()[0] >= 1: 71 return "Одновременно допускается не больше одного бронирования!" 72 73 # Проверим наличие книги и пользователя, а также предыдущие бронирования 74 cursor.execute(""" 75 SELECT b.name, COUNT(br.book_id) 76 FROM Books b 77 LEFT JOIN BookReservations br ON br.book_id = b.book_id AND br.reader_id = %s AND br.reservation_date IS NOT NULL 78 WHERE b.book_id = %s 79 GROUP BY b.name 80 """, (reader_id, book_id)) 81 result = cursor.fetchone() 82 83 if cursor.rowcount == 0 or result[1] > 0: 84 return None # Книга не найдена или уже забронирована пользователем 85 86 book_title = result[0] 87 88 # Создаем бронь в BookReservations 89 cursor.execute(""" 90 INSERT INTO BookReservations (name, book_id, reader_id, staff_id, reservation_date) 91 VALUES (%s, %s, %s, %s, CURRENT_DATE) 92 """, (book_title, book_id, reader_id, 0)) # Используйте фактический staff_id 93 reservation_id = cursor.lastrowid </pre>	<pre> 43 44 def get_available_books(): 45 """ 46 Получает список доступных книг с их авторами. 47 """ 48 query = """ 49 SELECT Books.book_id, Books.name, Authors.name 50 FROM Books 51 JOIN Authors ON Books.author_id = Authors.author_id 52 WHERE copies > 0 53 """ 54 rows = execute_query(query, fetchall=True) 55 available_books = [{'book_id': row[0], 'name': row[1], 'author': row[2]} for row in rows] 56 57 return available_books 58 59 def reserve_book(book_id, reader_id): 60 """ 61 Резервирует книгу для пользователя, создает бронь и запись о выдаче книги. 62 """ 63 # Проверим, есть ли у пользователя уже активные бронирования 64 query = """ 65 SELECT COUNT(*) FROM BookReservations 66 WHERE reader_id = %s AND reservation_date IS NOT NULL 67 """ 68 if execute_query(query, (reader_id,), fetchone=True)[0] >= 1: 69 return "Одновременно допускается не больше одного бронирования!" 70 71 # Проверим наличие книги и пользователя 72 query = """ 73 SELECT b.name, COUNT(br.book_id) 74 FROM Books b 75 LEFT JOIN BookReservations br ON br.book_id = b.book_id AND br.reader_id = %s AND br.reservation_date IS NOT NULL 76 WHERE b.book_id = %s 77 GROUP BY b.name 78 """ 79 result = execute_query(query, (reader_id, book_id), fetchone=True) 80 81 if not result or result[1] > 0: 82 return None # Книга не найдена или уже забронирована пользователем 83 84 book_title = result[0] 85 86 # Создаем бронь в BookReservations 87 query = """ 88 INSERT INTO BookReservations (name, book_id, reader_id, staff_id, reservation_date) 89 VALUES (%s, %s, %s, %s, CURRENT_DATE) 90 """ 91 execute_query(query, (book_title, book_id, reader_id, 1)) 92 </pre>
<pre> 93 reservation_id = cursor.lastrowid 94 95 # Создаем запись в BookLoans 96 return_date = datetime.date.today() + datetime.timedelta(days=14) 97 cursor.execute(""" 98 INSERT INTO BookLoans (book_id, reader_id, staff_id, issue_date, return_period) 99 VALUES (%s, %s, %s, CURRENT_DATE, %s) 100 """, (book_id, reader_id, 0, return_date)) # Используйте фактический staff_id 101 loan_id = cursor.lastrowid 102 103 # Уменьшаем количество доступных копий 104 cursor.execute("UPDATE Books SET copies = copies - 1 WHERE book_id = %s", (book_id,)) 105 conn.commit() 106 107 return reservation_id, loan_id, return_date 108 </pre>	<pre> 93 # Создаем запись в BookLoans 94 return_date = datetime.date.today() + datetime.timedelta(days=14) 95 query = """ 96 INSERT INTO BookLoans (book_id, reader_id, staff_id, issue_date, return_period) 97 VALUES (%s, %s, %s, CURRENT_DATE, %s) 98 """ 99 execute_query(query, (book_id, reader_id, 1, return_date)) 100 101 # Уменьшаем количество доступных копий 102 query = "UPDATE Books SET copies = copies - 1 WHERE book_id = %s" 103 execute_query(query, (book_id,)) 104 105 return "Книга успешно забронирована", return_date </pre>
<pre> 109 110 def get_user_reservations(reader_id): 111 conn = create_connection() 112 reservations = [] 113 with conn: 114 cursor = conn.cursor() 115 116 # Проверим, существует ли reader_id 117 cursor.execute(""" 118 SELECT br.reservation_id, b.name, br.reservation_date 119 FROM BookReservations br 120 JOIN Books b ON br.book_id = b.book_id 121 WHERE br.reader_id = %s 122 """, (reader_id,)) 123 rows = cursor.fetchall() 124 for row in rows: 125 reservations.append({'reservation_id': row[0], 'book_name': row[1], 'reservation_date': row[2]}) 126 127 return reservations </pre>	<pre> 109 110 def get_user_reservations(reader_id): 111 """ 112 Возвращает список текущих бронирований пользователя. 113 """ 114 query = """ 115 SELECT br.reservation_id, b.name, br.reservation_date 116 FROM BookReservations br 117 JOIN Books b ON br.book_id = b.book_id 118 WHERE br.reader_id = %s 119 """ 120 rows = execute_query(query, (reader_id,), fetchall=True) 121 reservations = [{'reservation_id': row[0], 'book_name': row[1], 'reservation_date': row[2]} for row in rows] 122 123 return reservations </pre>

<div>DatabaseInteractions/user_management.py</div> <div> <pre> 1 - from psycopg2 import extras 2 - from db import * 3 4 def register_user(name, contact_data, reader_number): 5 conn = create_connection() 6 with conn: 7 with conn.cursor() as cursor: 8 # Проверяем на существование пользователя с таким именем 9 cursor.execute("SELECT * FROM Readers WHERE name = %s", (name,)) 10 if cursor.fetchone(): 11 return "Такое имя пользователя уже зарегистрировано!" 12 13 # Перезаписываем пользователя 14 query = "INSERT INTO Readers(name, contact_data, reader_number) 15 VALUES(%s, %s, %s) RETURNING reader_id" 16 cursor.execute(query, (name, contact_data, reader_number)) 17 conn.commit() 18 return cursor.fetchone()[0] # Возвращает идентификатор нового пользователя 19 20 21 def log_in_user(name, reader_number): 22 conn = create_connection() 23 with conn: 24 with conn.cursor() as cursor: 25 query = "SELECT reader_id FROM Readers WHERE name = %s AND reader_number = %s" 26 cursor.execute(query, (name, reader_number)) 27 result = cursor.fetchone() 28 return result[0] if result else None 29 30 def log_in_staff(name): 31 conn = create_connection() 32 with conn: 33 with conn.cursor() as cursor: 34 query = "SELECT staff_id FROM LibraryStaff WHERE name = %s" 35 cursor.execute(query, (name,)) 36 result = cursor.fetchone() 37 return result[0] if result else None 38 39 40 def get_user_profile(user_id): 41 conn = create_connection() 42 with conn: 43 with conn.cursor(cursor_factory=psycopg2.extras.DictCursor) as cursor: 44 cursor.execute("SELECT name, contact_data, reader_number FROM Readers WHERE reader_id = %s", (user_id,)) 45 return cursor.fetchone() 46 47 def user_exists(user_id): 48 conn = create_connection() 49 with conn: 50 with conn.cursor() as cursor: 51 cursor.execute("SELECT 1 FROM Readers WHERE reader_id = %s", (user_id,)) 52 return cursor.fetchone() is not None </pre> </div>	<div> <pre> 1 + from Configuration.db_operations import * 2 + import psycopg2.extras 3 4 def register_user(name, contact_data, reader_number): 5 + 6 + Перезаписывает нового пользователя, если имя не существует в базе данных. 7 + 8 + Проверяем на существование пользователя с таким именем 9 + if check_if_exists("SELECT * FROM Readers WHERE name = %s", (name,)): 10 + return "Такое имя пользователя уже зарегистрировано!" 11 12 + 13 + # Перезаписываем пользователя 14 + query = "INSERT INTO Readers(name, contact_data, reader_number) 15 + VALUES(%s, %s, %s) RETURNING reader_id" 16 + return execute_query(query, (name, contact_data, reader_number), fetchone=True)[0] 17 18 + 19 + # Проверяет существование пользователя в базе данных по имени и номеру читателя. 20 + 21 + query = "SELECT reader_id FROM Readers WHERE name = %s AND reader_number = %s" 22 + result = execute_query(query, (name, reader_number), fetchone=True) 23 + return result[0] if result else None 24 25 def log_in_staff(name): 26 + 27 + Проверяет существование сотрудника в базе данных по имени. 28 + 29 + query = "SELECT staff_id FROM LibraryStaff WHERE name = %s" 30 + result = execute_query(query, (name,), fetchone=True) 31 + return result[0] if result else None 32 33 34 + 35 + Возвращает профиль пользователя по его ID. 36 + 37 + query = "SELECT name, contact_data, reader_number FROM Readers WHERE reader_id = %s" 38 + return execute_query(query, (user_id,), fetchone=True) 39 40 def user_exists(user_id): 41 + 42 + Проверяет, существует ли пользователь с данным ID. 43 + 44 + return check_if_exists("SELECT 1 FROM Readers WHERE reader_id = %s", (user_id,)) </pre> </div>
---	---

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

<div>Handlers/library_handlers.py</div> <div> <pre> 1 - from aiogram.fsm.context import FSMContext 2 - from aiogram.fsm.state import StatesGroup, State 3 4 - from DatabaseInteractions.admin_database_updates import delete_book, change_copies, add_book, delete_genre, add_genre, 5 - add_department, \ 6 - add_publisher, add_author, delete_author, delete_department, delete_publisher 7 - from Handlers.bot_handlers import router, ask_question 8 9 @-84,6 +85,7 @ async def book_isbn_entered(message: types.Message, state: FSMContext): 10 11 @router.message(AddBook.waiting_for_author_id) 12 13 async def author_id_entered(message: types.Message, state: FSMContext): 14 try: 15 author_id = int(message.text) 16 17 @-96,6 +98,7 @ async def author_id_entered(message: types.Message, state: FSMContext): 18 19 @router.message(AddBook.waiting_for_publisher_id) 20 21 async def publisher_id_entered(message: types.Message, state: FSMContext): 22 try: 23 publisher_id = int(message.text) 24 25 @-108,6 +111,7 @ async def publisher_id_entered(message: types.Message, state: FSMContext): 26 27 @router.message(AddBook.waiting_for_genre_id) 28 29 async def genre_id_entered(message: types.Message, state: FSMContext): 30 try: 31 genre_id = int(message.text) 32 33 @-120,6 +124,7 @ async def genre_id_entered(message: types.Message, state: FSMContext): 34 35 @router.message(AddBook.waiting_for_department_id) 36 37 async def department_id_entered(message: types.Message, state: FSMContext): 38 try: 39 department_id = int(message.text) </pre> </div>	<div> <pre> 1 + from aiogram.fsm.context import FSMContext 2 + from aiogram.fsm.state import StatesGroup, State 3 4 + from Configuration.db_operations import * 5 6 + from DatabaseInteractions.admin_database_updates import delete_book, change_copies, add_book, add_genre, add_department, \ 7 + add_publisher, add_author, delete_author, delete_department, delete_publisher 8 + from Handlers.bot_handlers import router, ask_question 9 10 11 + @handle_db_errors 12 + async def author_id_entered(message: types.Message, state: FSMContext): 13 try: 14 author_id = int(message.text) 15 16 17 + @handle_db_errors 18 + async def publisher_id_entered(message: types.Message, state: FSMContext): 19 try: 20 publisher_id = int(message.text) 21 22 23 + @handle_db_errors 24 + async def genre_id_entered(message: types.Message, state: FSMContext): 25 try: 26 genre_id = int(message.text) 27 28 29 + @handle_db_errors 30 + async def department_id_entered(message: types.Message, state: FSMContext): 31 try: 32 department_id = int(message.text) </pre> </div>
--	--

132		137	
133		138	
134	@router.message(AddBook.waiting_for_copies)	139	@router.message(AddBook.waiting_for_copies)
135	async def copies_entered(message: types.Message, state: FSMContext):	140	+ @handle_db_errors
136	try:	141	async def copies_entered(message: types.Message, state: FSMContext):
137	copies = int(message.text)	142	try:
138		143	copies = int(message.text)
139	@@ -151,6 +157,7 @@ async def copies_entered(message: types.Message, state: FSMContext):		
140			
141	await state.set_state(None)	157	await state.set_state(None)
142		158	
143		159	
144		160	+
145		161	# endregion
146	# endregion	162	
147	# region update amount	163	# region update amount
148	@@ -220,8 +227,8 @@ async def cmd_delete_book(message: types.Message, state: FSMContext):		
149			
150	await message.answer("Пожалуйста, войдите в систему как персонал для использования этой команды.")	227	await message.answer("Пожалуйста, войдите в систему как персонал для использования этой команды.")
151		228	
152		229	
153			
154	- # Function to handle the entered book ID for deletion	230	@router.message(DeleteBook.waiting_for_book_id)
155	@router.message(DeleteBook.waiting_for_book_id)	231	+ @handle_db_errors
156	async def delete_book_id_entered(message: types.Message, state: FSMContext):	232	async def delete_book_id_entered(message: types.Message, state: FSMContext):
157	try:	233	try:
158	book_id = int(message.text)	234	book_id = int(message.text)
159	@@ -231,7 +238,6 @@ async def delete_book_id_entered(message: types.Message, state: FSMContext):		
160			
161	return	238	return
162		239	
163	result = delete_book(book_id)	240	result = delete_book(book_id)
164			
165	-		
166	await message.answer(f"Книга с ID {book_id} успешно удалена.")	241	await message.answer(f"Книга с ID {book_id} успешно удалена.")
167	await state.set_state(None)	242	await state.set_state(None)
168		243	
169	@@ -276,18 +282,17 @@ async def cmd_delete_genre(message: types.Message, state: FSMContext):		
170			
171		282	
172		283	
173		284	
174	- # Function to handle the entered genre ID for deletion	285	@router.message(DeleteAuthor.waiting_for_author_id)
175	@router.message(DeleteGenre.waiting_for_genre_id)	286	+ @handle_db_errors
176	async def delete_genre_id_entered(message: types.Message, state: FSMContext):	287	async def delete_author_id_entered(message: types.Message, state: FSMContext):
177	try:	288	try:
178	genre_id = int(message.text)	289	author_id = int(message.text)
179	-	290	except ValueError:
180	except ValueError:	291	await message.answer("Пожалуйста, введите корректный ID автора (число).")
181	await message.answer("Пожалуйста, введите корректный ID жанра (число).")	292	
182	return	293	return
183			
184			
185			
186			
187			
188			
189			
190			
191			
192			
193			
194			
195			
196			
197			
198			
199			
200			
201			
202			
203			
204			
205			
206			
207			
208			
209			
210			
211			
212			
213			
214			
215			
216			
217			
218			
219			
220			
221			
222			
223			
224			
225			
226			
227			
228			
229			
230			
231			
232			
233			
234			
235			
236			
237			
238			
239			
240			
241			
242			
243			
244			
245			
246			
247			
248			
249			
250			
251			
252			
253			
254			
255			
256			
257			
258			
259			
260			
261			
262			
263			
264			
265			
266			
267			
268			
269			
270			
271			
272			
273			
274			
275			
276			
277			
278			
279			
280			
281			
282			
283			
284			
285			
286			
287			
288			
289			
290			
291			
292			
293			
294			
295			
296			
297			
298			
299			
300			
301			
302			
303			
304			
305			
306			
307			
308			
309			
310			
311			
312			
313			
314			
315			
316			
317			
318			
319			
320			
321			
322			
323			
324			
325			
326			
327			
328			
329			
330			
331			
332			
333			
334			
335			
336			
337			
338			
339			
340			
341			
342			
343			
344			
345			
346			
347			
348			
349			
350			
351			
352			
353			
354			
355			
356			
357			
358			
359			
360			
361			
362			
363			
364			
365			
366			
367			
368			
369			
370			
371			
372			
373			
374			
375			
376			
377			
378			
379			
380			
381			
382			
383			
384			
385			
386			
387			
388			
389			
390			
391			
392			
393			
394			
395			
396			
397			
398			
399			
400			
401			
402			
403			
404			
405			
406			
407			
408			
409			
410			
411			
412			
413			
414			
415			
416			
417			
418			
419			
420			
421			
422			
423			
424			
425			
426			
427			
428			
429			
430			
431			
432			
433			
434			
435			
436			
437			
438			
439			
440			
441			
442			
443			
444			
445			