Лабораторная работа 2. Автоматическое заполнение базы данных. Агрегатные функции

I. Наполнение БД через JSON или XML

1. Принципы автоматического заполнения БД

Вместо ручного ввода данных в SQLite можно использовать *автоматическое заполнение* с помощью файлов JSON или XML. Эти форматы удобны для хранения структурированных данных и позволяют легко загружать их в базу.

JSON (JavaScript Object Notation) легко читается и обрабатывается в Python через модуль `json`. Представляет данные в виде *ключ-значение*, аналогично структуре таблиц в БД. Поддерживается во многих языках программирования.

XML (Extensible Markup Language) использует *теги* для обозначения данных. Менее удобен в обработке, чем JSON, но хорошо структурирован. Обрабатывается в Python через модуль `xml.etree.ElementTree`.

Формат JSON удобнее для работы в Python, так как json.load() преобразует данные в словари и списки.

Формат XML более формализован, но требует парсинга[1] через ElementTree, что делает код сложнее.

Оба метода позволяют автоматически загружать данные в БД без ручного ввода.

2. Наполнение БД из JSON

Создадим файл data.json с данными для заполнения таблиц.

```
{
    "genres": [
        {"GenreID": 1, "GenreName": "роман", "Description": "Художественная
проза"},
        {"GenreID": 2, "GenreName": "драма", "Description": "Серьезное
литературное произведение"}
    ],
    "authors": [
     {"AuthorID": 1, "Name": "М. Горький", "BirthYear": 1868, "Nationality":
"русский"},
     {"AuthorID": 2, "Name": "А.Чехов", "BirthYear": 1860, "Nationality":
"русский"}
    ],
    "books": [
     {"Title": "Мать", "Author": 1, "Genre": 1, "PublicationYear": 2006,
"ISBN": "978-1234567890"},
     {"Title": "На дне", "Author": 1, "Genre": 2, "PublicationYear": 2002,
"ISBN": "978-0987654321"}
    ],
    "book_instances": [
     {"Book": 1, "VolumeNumber": 1, "VolumeStatus": "Доступен", "Condition":
"Хорошее"},
     {"Book": 2, "VolumeNumber": 1, "VolumeStatus": "Выдан", "Condition":
"Отличное"}
    ]
}
```

[1] Парсинг - процесс извлечения нужной информации из текстового или структурированного файла

Python-код для загрузки данных из JSON в SQLite:

```
import sqlite3
import json

# Подключение к базе данных
conn = sqlite3.connect("library_2a.db")
cursor = conn.cursor()

# Чтение JSON-файла
with open("data.json", "r", encoding="utf-8") as file:
    data = json.load(file)

# Заполнение таблицы Genre
for genre in data["genres"]:
    cursor.execute("INSERT INTO Genre (GenreID, GenreName, Description)
VALUES (?, ?, ?)", (genre["GenreID"], genre["GenreName"],
genre["Description"]))

# Заполнение таблицы Author
for author in data["authors"]:
    cursor.execute("INSERT INTO Author (AuthorID, Name, BirthYear,
Nationality) VALUES (?, ?, ?, ?)", (author["AuthorID"], author["Name"],
author["BirthYear"], author["Nationality"]))

# Заполнение таблицы Book
for book in data["books"]:
    cursor.execute("INSERT INTO Book (Title, Author, Genre, PublicationYear,
ISBN) VALUES (?, ?, ?, ?, ?)", (book["Title"], book["Author"], book["Genre"],
book["PublicationYear"], book["ISBN"]))

# Заполнение таблицы BookInstance
for instance in data["book_instances"]:
    cursor.execute("INSERT INTO BookInstance (Book, VolumeNumber,
VolumeStatus, Condition) VALUES (?, ?, ?, ?)", (instance["Book"],
instance["VolumeNumber"], instance["VolumeStatus"], instance["Condition"]))

# Сохранение изменений
conn.commit()
conn.close()

print("Данные из JSON-файла успешно загружены в базу!")
```

3. Наполнение БД из XML

Создадим XML-файл data.xml с аналогичными данными.

Пример XML-файла (data.xml):

```
<library>
    <genres>
        <genre>
            <GenreID>1</GenreID>
            <GenreName>роман</GenreName>
            <Description>Художественная проза</Description>
        </genre>
        <genre>
```

```
            <GenreID>2</GenreID>
            <GenreName>драма</GenreName>
            <Description>Серьезное литературное произведение</Description>
        </genre>
    </genres>

    <authors>
        <author>
            <AuthorID>1</AuthorID>
            <Name>М. Горький</Name>
            <BirthYear>1868</BirthYear>
            <Nationality>русский</Nationality>
        </author>
        <author>
            <AuthorID>2</AuthorID>
            <Name>А. Чехов</Name>
            <BirthYear>1860</BirthYear>
            <Nationality>русский</Nationality>
        </author>
    </authors>

    <books>
        <book>
            <Title>Мать</Title>
            <Author>1</Author>
            <Genre>1</Genre>
            <PublicationYear>2006</PublicationYear>
            <ISBN>978-1234567890</ISBN>
        </book>
        <book>
            <Title>На дне</Title>
            <Author>1</Author>
            <Genre>2</Genre>
            <PublicationYear>2002</PublicationYear>
            <ISBN>978-0987654321</ISBN>
        </book>
    </books>

    <book_instances>
        <book_instance>
            <Book>1</Book>
            <VolumeNumber>1</VolumeNumber>
            <VolumeStatus>Доступен</VolumeStatus>
            <Condition>Хорошее</Condition>
        </book_instance>
        <book_instance>
            <Book>2</Book>
            <VolumeNumber>1</VolumeNumber>
            <VolumeStatus>Выдан</VolumeStatus>
            <Condition>Отличное</Condition>
        </book_instance>
    </book_instances>
</library>
```

Python-код для загрузки данных из XML в SQLite:

```
import sqlite3
import xml.etree.ElementTree as ET

# Подключение к базе данных
conn = sqlite3.connect("library_2b.db")
cursor = conn.cursor()

# Чтение XML-файла
tree = ET.parse("data.xml")  # Убедись, что у тебя есть этот файл
root = tree.getroot()

# Функция для вставки данных
def insert_data(table, columns, values):
    placeholders = ", ".join("?" * len(values))
    query = f"INSERT INTO {table} ({', '.join(columns)}) VALUES
({placeholders})"
    cursor.execute(query, values)

# Заполняем таблицу Genre
for genre in root.find("genres").findall("genre"):
    insert_data("Genre", ["GenreID", "GenreName", "Description"], [
        int(genre.find("GenreID").text),
        genre.find("GenreName").text,
        genre.find("Description").text
    ])

# Заполняем таблицу Author
for author in root.find("authors").findall("author"):
    insert_data("Author", ["AuthorID", "Name", "BirthYear", "Nationality"], [
        int(author.find("AuthorID").text),
        author.find("Name").text,
        int(author.find("BirthYear").text),
        author.find("Nationality").text
    ])

# Заполняем таблицу Book
for book in root.find("books").findall("book"):
    insert_data("Book", ["Title", "Author", "Genre", "PublicationYear",
"ISBN"], [
        book.find("Title").text,
        int(book.find("Author").text),
        int(book.find("Genre").text),
        int(book.find("PublicationYear").text),
        book.find("ISBN").text
    ])

# Заполняем таблицу BookInstance
for instance in root.find("book_instances").findall("book_instance"):
    insert_data("BookInstance", ["Book", "VolumeNumber", "VolumeStatus",
"Condition"], [
        int(instance.find("Book").text),
        int(instance.find("VolumeNumber").text),
        instance.find("VolumeStatus").text,
        instance.find("Condition").text
    ])
```

```
# Сохранение изменений и закрытие соединения
conn.commit()
conn.close()

print("Данные из XML-файла успешно загружены в базу!")
```

4. Функция insert_data()

Функция insert_data() предназначена для вставки данных в SQLite-таблицы. Разберем её подробно.

```
def insert_data(table, columns, values):
    placeholders = ', '.join(['?'] * len(values))
    query = f"INSERT OR IGNORE INTO {table} ({', '.join(columns)}) VALUES
({placeholders})"
    cursor.execute(query, values)
```

1. Аргументы функции

table – название таблицы, в которую вносятся данные (например, "Book").

columns – список имен столбцов, куда вставляем значения (например, ["Title", "Author", "Genre", "PublicationYear", "ISBN"]).

values – список значений для вставки в эти столбцы (например, ["Мать", 1, 1, 2006, "978-1234567890"]).

2. Создание плейсхолдеров[2] для значений

```
placeholders = ', '.join(['?'] * len(values))
```

Генерируем строку, состоящую из символов "?", соответствующую количеству значений.

Например, если values = ["Мать", 1, 1, 2006, "978-1234567890"], то len(values) = 5, и получится строка: ?, ?, ?, ?, ?

Она нужна для безопасной подстановки значений в SQL-запрос.

3. Формирование SQL-запроса

```
query = f"INSERT OR IGNORE INTO {table} ({', '.join(columns)}) VALUES
({placeholders})"
```

Вставляем в SQL-запрос: имя таблицы (table); список столбцов (columns), объединенный через "," ; подставляем "?" как плейсхолдеры для значений.

Пример сформированного SQL-запроса:

```
INSERT OR IGNORE INTO Book (Title, Author, Genre, PublicationYear, ISBN)
VALUES (?, ?, ?, ?, ?)
```

---

[2] Плейсхолдер (от англ. *placeholder*) — это заместитель или заглушка. В программировании плейсхолдер – это специальный символ или конструкция, которая временно занимает место для какого-то значения и затем заменяется реальными данными.

Модифицированный вариант SQL-команды INSERT – "INSERT OR IGNORE" означает: вставить запись, если такой записи еще нет, но игнорирует ошибку**,** если происходит нарушение уникальных ограничений, например, если уже существует запись с таким же `PRIMARY KEY` или `UNIQUE` значением. В таком случае запись не вставляется.

4. Выполнение запроса

```
cursor.execute(query, values)
```

Метод cursor.execute() выполняет SQL-запрос, подставляя значения в ?, что защищает от SQL-инъекций. Например, если values = ["Мать", 1, 1, 2006, "978-1234567890"], то запрос на выполнение будет:

```
INSERT OR IGNORE INTO Book (Title, Author, Genre, PublicationYear, ISBN)
VALUES ('Мать', 1, 1, 2006, '978-1234567890')
```

Преимущества функции:

- Гибкость – можно вставлять данные в любую таблицу SQLite.
- Безопасность – "?" предотвращает SQL-инъекции.
- Автоматизация – избавляет от написания множества INSERT INTO.
- Предотвращение дубликатов – OR IGNORE не дает вставлять повторяющиеся строки.

5. Добавление данных в уже существующую БД

В приведенных примерах мы просто вставляем новые данные, поэтому не добавляем логику поиска и обновления. Если бы в БД уже были данные, перед вставкой нужно было бы:

1. Сначала добавить авторов, потому что книги ссылаются на авторов, и важно, чтобы все авторы были в базе данных до вставки книг. Перед этим проверить, существует ли уже автор с таким именем и годом рождения в БД.
2. Затем вставить книги, но перед этим проверить, существует ли уже книга с таким же названием в базе данных (по полю `Title`), чтобы не вставлять дубликаты. Одновременно на этом шаге найти ID автора и связать книгу с автором.
3. Перед вставкой экземпляров книги необходимо найти соответствующую книгу и извлечь её `BookID` из базы данных и использовать его для связи между экземпляром и книгой.

Для выполнения этих шагов нужно использовать команды вроде `INSERT OR IGNORE` или `REPLACE INTO`, чтобы избежать добавления дубликатов.

Кроме того, в JSON для каждой книги нужно указать имя автора или другие данные, по которым можно найти автора в базе данных (например, имя и год рождения). А затем, в процессе загрузки данных в базу, искать этого автора в базе данных и использовать его `AuthorID`.

При вставке данных экземпляров одной и той же книги может быть очень много. Поэтому, чтобы не искать `BookID` для одной и той же книги каждый раз, лучше сохранить найденные ID в словаре[3]. Так запрос в БД будет осуществляться только один раз для каждой книги.

[3] Словарь в Python – это структура данных, которая хранит пары "ключ-значение". Ключом будет название книги, а Значением будет ID, который найден в БД (BookID).

6. Агрегатные функции

Агрегатные функции SQL помогают анализировать и обрабатывать данные, выявлять тенденции и генерировать сводную информацию из базы данных:

- `COUNT()`: подсчитывает количество строк в таблице или количество записей для определённого условия;
- `MIN()/ MAX()`: находит минимальное/максимальное значение в столбце;
- `AVG()`: вычисляет среднее значение в столбце;
- `SUM()`: вычисляет сумму значений в столбце;
- `GROUP_CONCAT()`: объединяет значения из нескольких строк в одну строк.

С помощью этих функций можно быстро получать основные метрики и выявлять важные характеристики данных, такие как общее количество записей, минимальные или максимальные значения и средние показатели.

Для подсчёта количества авторов и книг в базе данных можно использовать SQL-запросы для получения количества строк в таблицах `Author` и `Book`. Также, можно вычислить среднее значение года публикации книг, найти максимальный или минимальный год, или даже посчитать, сколько книг написаны каждым автором.

```
# Подсчёт количества авторов
cursor.execute("SELECT COUNT(*) FROM Author")
author_count = cursor.fetchone()[0]
print(f"Количество авторов: {author_count}")


# Подсчёт количества книг
cursor.execute("SELECT COUNT(*) FROM Book")
book_count = cursor.fetchone()[0]
print(f"Количество книг: {book_count}")


# Вычисление среднего года публикации книг
cursor.execute("SELECT AVG(PublicationYear) FROM Book")
avg_year = cursor.fetchone()[0]
print(f"Средний год публикации книг: {avg_year}")


# Нахождение минимального года публикации
cursor.execute("SELECT MIN(PublicationYear) FROM Book")
min_year = cursor.fetchone()[0]
print(f"Минимальный год публикации: {min_year}")


# Нахождение максимального года публикации
cursor.execute("SELECT MAX(PublicationYear) FROM Book")
max_year = cursor.fetchone()[0]
print(f"Максимальный год публикации: {max_year}")
```

V. Окончательный базовый код программы:

Программа_1. Заполнение БД из JSON (файл LAB_BASE_2a.py)

```
import sqlite3
import os
import json

# Путь к файлу базы данных SQLite
db_file = 'library_2a.db'

# ***************************
# эта команда нужна для отладки (для повторных запусков)
# Если файл БД уже существует, то удалить существующий файл БД,
# что бы предотвратить повторную запись тех же самых данных
if  os.path.isfile(db_file):
  os.remove(db_file)
# ***************************

# Проверка существования файла базы данных
if not os.path.isfile(db_file):
    # Если файл базы данных не существует, создаем его и подключаемся
    conn = sqlite3.connect(db_file)

    # Создание объекта курсора для выполнения SQL-запросов
    cursor = conn.cursor()

    # Создание таблиц и внесение данных в базу данных

    # Создание таблицы Genre
    cursor.execute('''CREATE TABLE IF NOT EXISTS Genre (
        GenreID INTEGER PRIMARY KEY,
        GenreName TEXT NOT NULL UNIQUE,
        Description TEXT
    )''')

    # Создание таблицы Author
    cursor.execute('''CREATE TABLE IF NOT EXISTS Author (
        AuthorID INTEGER PRIMARY KEY,
        Name TEXT NOT NULL,
        BirthYear INTEGER CHECK (BirthYear <= 2010),
        Nationality TEXT
    )''')

    # Создание таблицы Book
    cursor.execute('''CREATE TABLE IF NOT EXISTS Book (
        BookID INTEGER PRIMARY KEY,
        Author INTEGER NOT NULL,
        Genre INTEGER NOT NULL,
        Title TEXT NOT NULL,
```

```
        PublicationYear INTEGER NOT NULL,
        ISBN TEXT UNIQUE,
        FOREIGN KEY (Author) REFERENCES Author(AuthorID),
        FOREIGN KEY (Genre) REFERENCES Genre(GenreID)
    )''')

    # Создание таблицы BookInstance
    cursor.execute('''CREATE TABLE IF NOT EXISTS BookInstance (
        BookInstanceID INTEGER PRIMARY KEY,
        Book INTEGER NOT NULL,
        VolumeNumber INTEGER NOT NULL DEFAULT 1,
        VolumeStatus TEXT,
        Condition TEXT,
        FOREIGN KEY (Book) REFERENCES Book(BookID)
    )''')

    # Сохранение изменений
    conn.commit()


else:
    # Если файл базы данных существует, просто подключаемся к нему
    conn = sqlite3.connect(db_file)
    # и создаем объект курсор для выполнения SQL-запросов
    cursor = conn.cursor()

# Чтение JSON-файла
with open("data.json", "r", encoding="utf-8") as file:
    data = json.load(file)

# Заполнение таблицы Genre
for genre in data["genres"]:
    cursor.execute("INSERT INTO Genre (GenreID, GenreName, Description) VALUES
(?, ?, ?)", (genre["GenreID"], genre["GenreName"], genre["Description"]))

# Заполнение таблицы Author
for author in data["authors"]:
    cursor.execute("INSERT INTO Author (AuthorID, Name, BirthYear, Nationality)
VALUES (?, ?, ?, ?)", (author["AuthorID"], author["Name"], author["BirthYear"],
author["Nationality"]))

# Заполнение таблицы Book
for book in data["books"]:
    cursor.execute("INSERT INTO Book (Title, Author, Genre, PublicationYear,
ISBN) VALUES (?, ?, ?, ?, ?)", (book["Title"], book["Author"], book["Genre"],
book["PublicationYear"], book["ISBN"]))

# Заполнение таблицы BookInstance
for instance in data["book_instances"]:
```

```
    cursor.execute("INSERT INTO BookInstance (Book, VolumeNumber, VolumeStatus,
Condition) VALUES (?, ?, ?, ?)", (instance["Book"], instance["VolumeNumber"],
instance["VolumeStatus"], instance["Condition"]))

# Сохранение изменений
conn.commit()


print("Данные из JSON-файла успешно загружены в базу!")


# Получение результатов запроса и вывод на экран
cursor.execute("SELECT * FROM Book")
results = cursor.fetchall()

# Обработка результатов
if results:  # Проверяем, есть ли данные
    for row in results:
        print(row)
else:
    print("Нет данных для вывода.")



# Подсчёт количества книг
cursor.execute("SELECT COUNT(*) FROM Book")
book_count = cursor.fetchone()[0]
print(f"Количество книг: {book_count}")

# Вычисление среднего года публикации книг
cursor.execute("SELECT AVG(PublicationYear) FROM Book")
avg_year = cursor.fetchone()[0]
print(f"Средний год публикации книг: {round(avg_year)}")


# Нахождение минимального года публикации
cursor.execute("SELECT MIN(PublicationYear) FROM Book")
min_year = cursor.fetchone()[0]
print(f"Минимальный год публикации: {min_year}")


# Нахождение максимального года публикации
cursor.execute("SELECT MAX(PublicationYear) FROM Book")
max_year = cursor.fetchone()[0]
print(f"Максимальный год публикации: {max_year}")

# Не забудьте закрыть соединение с базой данных, когда закончите работу

conn.close()
```

Результаты работы программы_1:

```
Данные из JSON-файла успешно загружены в базу!
(1, 1, 1, 'Мать', 2006, '978-1234567890')
(2, 1, 2, 'На дне', 2002, '978-0987654321')
Количество книг: 2
Средний год публикации книг: 2004
Минимальный год публикации: 2002
Максимальный год публикации: 2006
```

Программа_2 Автозаполнение из XML-файла (файл LAB_BASE_2b.py)

```
import sqlite3
import os
import  xml.etree.ElementTree as ET

# Путь к файлу базы данных SQLite
db_file = 'library_2b.db'

# ***************************
# эта команда нужна для отладки (для повторных запусков)
# Если файл БД уже существует, то удалить существующий файл БД,
# что бы предотвратить повторную запись тех же самых данных
if  os.path.isfile(db_file):
  os.remove(db_file)
# ***************************

# Проверка существования файла базы данных
if not os.path.isfile(db_file):
    # Если файл базы данных не существует, создаем его и подключаемся
    conn = sqlite3.connect(db_file)

    # Создание объекта курсора для выполнения SQL-запросов
    cursor = conn.cursor()

    # Создание таблиц и внесение данных в базу данных

    # Создание таблицы Genre
    cursor.execute('''CREATE TABLE IF NOT EXISTS Genre (
        GenreID INTEGER PRIMARY KEY,
        GenreName TEXT NOT NULL UNIQUE,
        Description TEXT
    )''')

    # Создание таблицы Author
    cursor.execute('''CREATE TABLE IF NOT EXISTS Author (
        AuthorID INTEGER PRIMARY KEY,
        Name TEXT NOT NULL,
```

```
        BirthYear INTEGER CHECK (BirthYear <= 2010),
        Nationality TEXT
    )''')

    # Создание таблицы Book
    cursor.execute('''CREATE TABLE IF NOT EXISTS Book (
        BookID INTEGER PRIMARY KEY,
        Author INTEGER NOT NULL,
        Genre INTEGER NOT NULL,
        Title TEXT NOT NULL,
        PublicationYear INTEGER NOT NULL,
        ISBN TEXT UNIQUE,
        FOREIGN KEY (Author) REFERENCES Author(AuthorID),
        FOREIGN KEY (Genre) REFERENCES Genre(GenreID)
    )''')

    # Создание таблицы BookInstance
    cursor.execute('''CREATE TABLE IF NOT EXISTS BookInstance (
        BookInstanceID INTEGER PRIMARY KEY,
        Book INTEGER NOT NULL,
        VolumeNumber INTEGER NOT NULL DEFAULT 1,
        VolumeStatus TEXT,
        Condition TEXT,
        FOREIGN KEY (Book) REFERENCES Book(BookID)
    )''')

    # Сохранение изменений
    conn.commit()


else:
    # Если файл базы данных существует, просто подключаемся к нему
    conn = sqlite3.connect(db_file)
    # и создаем объект курсор для выполнения SQL-запросов
    cursor = conn.cursor()

# Чтение XML-файла
tree = ET.parse("data.xml")  # Убедись, что у тебя есть этот файл
root = tree.getroot()

# Функция для вставки данных
def insert_data(table, columns, values):
    placeholders = ", ".join("?" * len(values))
    query = f"INSERT INTO {table} ({', '.join(columns)}) VALUES ({placeholders})"
    cursor.execute(query, values)

# Заполняем таблицу Genre
for genre in root.find("genres").findall("genre"):
    insert_data("Genre", ["GenreID", "GenreName", "Description"], [
        int(genre.find("GenreID").text),
```

```
        genre.find("GenreName").text,
        genre.find("Description").text
    ])

# Заполняем таблицу Author
for author in root.find("authors").findall("author"):
    insert_data("Author", ["AuthorID", "Name", "BirthYear", "Nationality"], [
        int(author.find("AuthorID").text),
        author.find("Name").text,
        int(author.find("BirthYear").text),
        author.find("Nationality").text
    ])

# Заполняем таблицу Book
for book in root.find("books").findall("book"):
    insert_data("Book", ["Title", "Author", "Genre", "PublicationYear", "ISBN"],
[
        book.find("Title").text,
        int(book.find("Author").text),
        int(book.find("Genre").text),
        int(book.find("PublicationYear").text),
        book.find("ISBN").text
    ])

# Заполняем таблицу BookInstance
for instance in root.find("book_instances").findall("book_instance"):
    insert_data("BookInstance", ["Book", "VolumeNumber", "VolumeStatus",
"Condition"], [
        int(instance.find("Book").text),
        int(instance.find("VolumeNumber").text),
        instance.find("VolumeStatus").text,
        instance.find("Condition").text
    ])


# Сохранение изменений
conn.commit()


print("Данные из XML-файла успешно загружены в базу!")


# Получение результатов запроса и вывод на экран
cursor.execute("SELECT * FROM Author")
results = cursor.fetchall()

# Обработка результатов
if results:  # Проверяем, есть ли данные
    for row in results:
        print(row)
else:
```

```
    print("Нет данных для вывода.")




# Подсчёт количества авторов
cursor.execute("SELECT COUNT(*) FROM Author")
author_count = cursor.fetchone()[0]
print(f"Количество авторов: {author_count}")



# Вычисление среднего года рождения авторов
cursor.execute("SELECT AVG(BirthYear) FROM Author")
avg_year = cursor.fetchone()[0]
print(f"Средний год рождения авторов: {round(avg_year)}")


# Нахождение самого молодого года рождения
cursor.execute("SELECT MIN(BirthYear) FROM Author")
youngest_year = cursor.fetchone()[0]

if youngest_year is not None:
    # Получение всех авторов с самым молодым годом рождения
    cursor.execute("""
        SELECT Name, BirthYear
        FROM Author
        WHERE BirthYear = ?
    """, (youngest_year,))

    youngest_authors = cursor.fetchall()

    print("Самые молодые авторы:")
    for author in youngest_authors:
        name, birth_year = author
        print(f"{name}, Год рождения: {birth_year}")
else:
    print("Не найдено авторов.")

# Не забудьте закрыть соединение с базой данных, когда закончите работу
conn.close()
```

Результаты работы программы_2:

```
Данные из XML-файла успешно загружены в базу!
(1, 'М. Горький', 1868, 'русский')
(2, 'А. Чехов', 1860, 'русский')
(3, 'С. Платонов', 1860, 'русский')
Количество авторов: 3
Средний год рождения авторов: 1863
Самые молодые авторы:
А. Чехов, Год рождения: 1860
С. Платонов, Год рождения: 1860
```

Задание на самостоятельную работу

1. Создать собственные JSON- и XML-файлы.
2. Запустить базовые программы и получить результаты запросов для извлечения данных из таблиц Book и Author.
3. Внести изменения в JSON и XML-файлы: добавить новых авторов и новые книги.
4. Написать SQL-запросы с использованием агрегатных функций.
5. По желанию студента разработать код для автоматического заполнения уже существующей БД. В коде необходимо предусмотреть механизмы проверки на наличие дублирующихся данных в базе.