Implementation of Database Management using SQLite in Python

Ahmad Farhan AlShammari
Department of Computer and Information Systems
College of Business Studies, PAAET
Kuwait

ABSTRACT
The goal of this research is to implement database management using SQLite in Python. Database management is used to organize, store, and manipulate data efficiently. It is done using a special software, called the "database management system" (DBMS), like SQLite. The purpose of database management is to ensure that data is always accessible, accurate, consistent, and secure. It also provides a solid foundation for data analysis and decision making.

The basic operations of database management using SQLite are explained: importing library, creating database, creating tables, adding data, retrieving data, printing data, searching data, ordering data, updating data, computing data, aggregating data, grouping data, deleting data, and dropping tables.

The developed program was tested on an experimental data. The program has successfully performed the basic operations of database management using SQLite and provided the required results.

Keywords
Computer Science, Artificial Intelligence, Machine Learning, Database Management, Database, DB, Structured Query Language, SQL, SQLite, Python, Programming.

1. INTRODUCTION
In the recent years, machine learning has played a major role in the development of computer systems. Machine learning (ML) is a branch of Artificial Intelligence (AI) which is focused on developing new models and algorithms to improve the performance and efficiency of computer programs [1-10].

Database management is very important in the field of machine learning. It provides "organized" data to the machine learning models. Then, they can process data, analyze features, and make predictions.

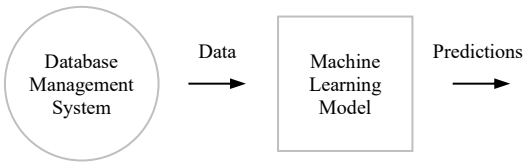

Fig 1: Importance of Database Management

Database management is used to organize, store, and manipulate data efficiently. It helps to keep data accessible, accurate, consistent, and secure. The database systems are applied everywhere in life; government, companies, banks, markets, hospitals, schools, universities, websites, search engines, etc.

2. LITERATURE REVIEW
The literature was reviewed to explore the fundamental concepts, methods, and applications of database management using SQLite [11-17, 18-25, 26-35].

Database management is the cornerstone of the information age. It is the core concept of computer systems because "data" is considered the most valuable asset in the organization.

The early database management systems (DBMS) were developed in the 1960s. For example, the "hierarchical" and the "network" models. Then, the "relational model" was proposed by Codd [36] in 1970. He suggested organizing data into tables with rows and columns. After that, the "Structured Query Language" (SQL) was developed by Chamberlin and Boyce in 1974 at IBM [37].

Since that time, many database management systems have been developed like DB2, Oracle, MySQL, SQLite, etc. Actually, the relational model and SQL are still forming the "backbone" of database systems in the world.

Now, with the evolution of internet and web technologies, new concepts have emerged like: NoSQL, big data, cloud computing, and data mining.

The fundamental concepts of database management using SQLite are explained in the following section.

Database Management:
Database management is the process of organizing, storing, and manipulating data efficiently. It is done using a special software, called the "database management system" (DBMS).

The purpose of database management is to ensure that data is always accessible, accurate, consistent, and secure.

The concept of database management is illustrated in the following diagram:

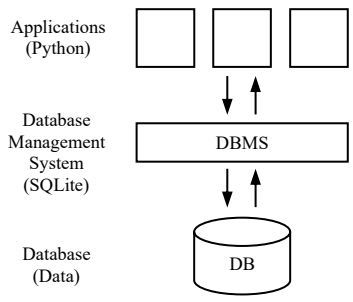

Fig 2: Concept of Database Management

Database (DB):

Database (DB) is a collection of organized data. The data is organized into tables with rows and columns. The row represents a single element, for example; a specific student. The column represents a piece of data, for example; student-no, name, and major.

Every table has a "primary key" to help in accessing data in the table. The table can have "foreign keys" to help in connecting with the other tables.

The concept of database is illustrated in the following diagram:

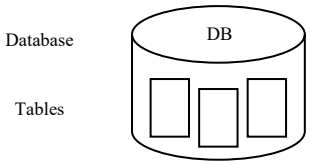

Fig 3: Concept of Database

The table structure is shown in the following diagram:

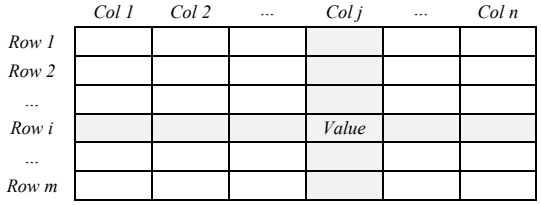

Fig 4: Structure of Table

The connections between tables are established using the foreign keys as shown in the following diagram:

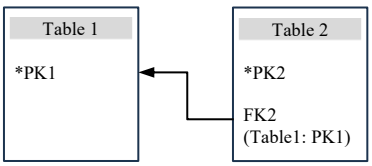

Fig 5: Connection between Tables

Here, the foreign key (*FK2*) in *Table 2* is connected to the primary key (*PK1*) in *Table 1*.

Example:

Assume the *College* database with the following tables: *Student*, *Course*, and *Enrollment*.

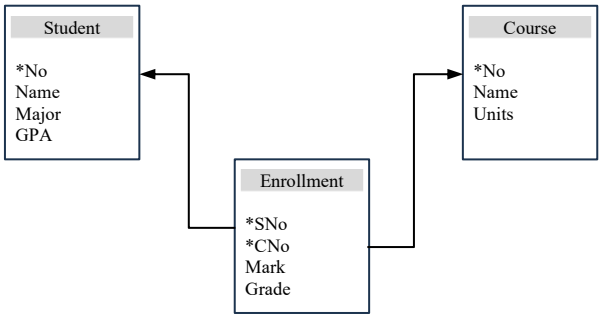

Fig 6: Example of Database

The diagram shows that the foreign keys (*SNo*) and (*CNo*) in table *Enrollment* are "referenced" to the primary keys in tables *Student* and *Course*, respectively.

Database Operations:

The basic operations of database are shown here in the list:

- Create Database
- Create Table
- Insert Row
- Update Row
- Delete Row
- Select Column
- Search Rows by Condition
- Order Rows by Column
- Group Rows by Column
- Aggregate Column
- Drop Table

Structured Query Language (SQL):

Structured Query Language (SQL) is a specific language used to define, manipulate, and control data in the database.

In general, the SQL commands are divided into four types:

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Data Query Language (DQL)
4. Data Control Language (DCL)

They are illustrated in the following diagram:

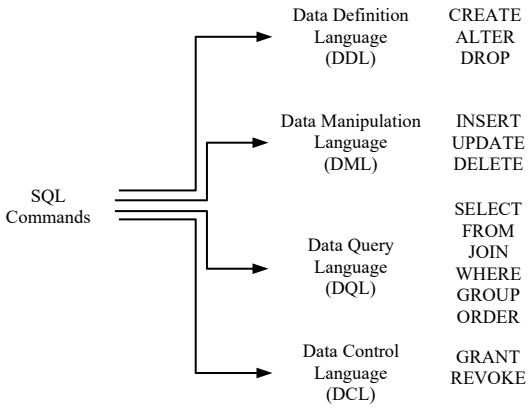

Fig 7: Types of SQL Commands

1. The DDL commands are used to define the structure of the database. For example:

- CREATE
- ALTER
- DROP

The DDL commands are shown in the following view:

```
CREATE DATABASE database;

CREATE TABLE table
(column   type,
 ...
 PRIMARY KEY(column, ...),
 FOREIGN KEY(column) REFERENCES table(column));
```

```
ALTER TABLE table ...;

DROP TABLE table;
```

2. The DML commands are used to manipulate (add, update, and delete) data in the database. For example:

- INSERT
- UPDATE
- DELETE

The DML commands are shown in the following view:

```
INSERT INTO table
(column, ...) VALUES (value, ...);

UPDATE table
SET column = value
WHERE condition;

DELETE FROM table
WHERE condition;
```

3. The DQL commands are used to perform queries (retrieve, search, group, and order) on the database. For example:

- SELECT
- FROM
- JOIN
- WHERE
- GROUP
- ORDER

The DQL commands are shown in the following view:

```
SELECT column, ...
FROM table, ...
JOIN table ON ...
WHERE condition
GROUP BY coulumn, ...
ORDER BY column ASC/DESC ...;
```

4. The DCL commands are used to control the user access and data security in the database. For example:

- GRANT
- REVOKE

The DCL commands are shown in the following view:

```
GRANT ...;

REVOKE ...;
```

SQLite:

SQLite [38] is an open-source relational database management system (RDBMS). It was developed by Richard Hipp in 2000. It has many advantages; it is small, fast, self-contained, and serverless. It is embedded by different programming languages, operating systems, and web browsers. SQLite is the most widely used database engine in the world.

Python:

Python [39] is an open-source, object-oriented, and general-purpose programming language. It is simple, easy to learn, and powerful. It is the most popular programming language, especially in the field of machine learning.

Python provides many additional libraries for different purposes. For example: Numpy [40], Pandas [41], Matplotlib [42], Seaborn [43], SciPy [44], NLTK [45], and SK Learn [46].

3. RESEARCH METHODOLOGY

The basic operations of database management using SQLite are: (1) importing library, (2) creating database, (3) creating tables, (4) adding data, (5) retrieving data, (6) printing data, (7) searching data, (8) ordering data, (9) updating data, (10) computing data, (11) aggregating data, (12) grouping data, (13) deleting data, and (14) dropping tables.

- Importing Library
- Creating Database
- Creating Tables
- Adding Data
- Retrieving Data
- Printing Data
- Searching Data
- Ordering Data
- Updating Data
- Computing Data
- Aggregating Data
- Grouping Data
- Deleting Data
- Dropping Tables

Fig 8: Operations of Database Management

The implementation of database management operations using SQLite is summarized in the following table:

Table 1: Implementation of Database Management Operations

Operation	Implementation
Import Library	`import sqlite3`
Create Database	`con = sqlite3.connect(…)`

Create Table	`CREATE TABLE table`
	`(col type PRIMARY KEY,`
	`…);`

Add Data	`INSERT INTO table` `(col, …) VALUES (value, …);`
Retrieve Data	`SELECT *` `FROM table;`

Print Data	`rows = …`
	`for row in rows:`
	`  print(row)`

Search Data	`SELECT *`
	`FROM table`
	`WHERE condition;`

Order Data	`SELECT *`
	`FROM table`
	`ORDER BY col ASC/DESC;`

Update Data	`value = input(…)`
	`UPDATE table`
	`SET col = value`
	`WHERE condition;`

Compute Data	`value = compute(…)`
	`UPDATE table`
	`SET col = value`
	`WHERE condition;`

Aggregate Data	`SELECT COUNT, AVG, MIN, MAX` `FROM table;`

Group Data	`SELECT …`
	`FROM table`
	`GROUP BY col;`

Delete Data	`DELETE FROM table` `WHERE condition;`
Drop Table	`DROP TABLE table;`

The basic operations of database management using SQLite are explained in the following section.

1. Importing Library:

The SQLite library (*sqlite3*) is imported by the following code:

```
import sqlite3
```

2. Creating Database:

The database (*database*) is created and the connection (*con*) is established by the following code:

```
def connect(database):
    con = sqlite3.connect(database)
    return con
```

3. Creating Tables:

The student table (*student*) is created by the following code:

```
def create_student():
    qry = """CREATE TABLE student
             (no    INTEGER PRIMARY KEY,
              name  TEXT,
              major TEXT,
              gpa   REAL)"""
    con.execute(qry)
    con.commit()
```

The course table (*course*) is created by the following code:

```
def create_course():
    qry = """CREATE TABLE course
             (no    INTEGER PRIMARY KEY,
              name  TEXT,
              units INTEGER)"""
    con.execute(qry)
    con.commit()
```

The enrollment table (*enroll*) is created by the following code:

```
def create_enroll():
    qry = """CREATE TABLE enroll
             (sno   INTEGER,
              cno   INTEGER,
              mark  INTEGER,
              grade TEXT,
              PRIMARY KEY(sno, cno),
              FOREIGN KEY(sno) REFERENCES
                               student(no),
              FOREIGN KEY(cno) REFERENCES
                               course(no))"""
    con.execute(qry)
    con.commit()
```

4. Adding Data:

The students are added to the table (*student*) by the user, one by one, as shown in the following code:

```
def add_student():
    print("Enter student-no or press <Enter> to
           quit ...\n")
    while (True):
        no = input("   No: ")
        if (not no):
            break
        else:
            no = int(no)
            name = input(" Name: ")
            major = input("Major: ")
            qry = """INSERT INTO student
                     (no, name, major, gpa)
                     VALUES (?, ?, ?, 0)"""
            cur.execute(qry, (no, name, major))
            con.commit()
            print()
```

The courses are added to the table (*course*) by the user, one by one, as shown in the following code:

```
def add_course():
    print("Enter course-no or press <Enter> to
           quit ...\n")
    while (True):
        no = input("   No: ")
        if (not no):
            break
        else:
            no = int(no)
            name = input(" Name: ")
            units = int(input("Units: "))
            qry = """INSERT INTO course
                     (no, name, units)
                     VALUES (?, ?, ?)"""
            cur.execute(qry, (no, name, units))
            con.commit()
            print()
```

The students in a group are added to the table (*enroll*) by the user, one by one, as shown in the following code:

```
def add_group(cno):
    print("Enter student-no or press <Enter> to
           quit ...\n")
    i = 0
    while (True):
        sno = input(f"{i+1}. Student No: ")
        if (not sno):
            break
        else:
            i += 1
            sno = int(sno)
            qry = """INSERT INTO enroll
                     (sno, cno, mark, grade)
                     VALUES (?, ?, 0, '')"""
            con.execute(qry, (sno, cno))
            con.commit()
```

5. Retrieving Data:

The data is retrieved from the table (*student*) by the following code:

```
def get_students():
    qry = "SELECT * FROM student"
    rows = con.execute(qry)
```

```
    return rows
```

The data is retrieved from the table (*course*) by the following code:

```
def get_courses():
    qry = "SELECT * FROM course"
    rows = con.execute(qry)
    return rows
```

The data is retrieved from the table (*enroll*) by the following code:

```
def get_enrolls():
    qry = "SELECT * FROM enroll"
    rows = con.execute(qry)
    return rows
```

6. Printing Data:

The data of students is printed by the following code:

```
def print_student(rows):
    print(f"{'No':3} {'Name':15} {'Major':15}
            {'GPA':4}")
    print("-"*40)
    for row in rows:
        no, name, major, gpa = row
        print(f"{no:3} {name:15} {major:15}
                {gpa:4.2f}")
    print()
```

The data of courses is printed by the following code:

```
def print_course(rows):
    print(f"{'No':3} {'Name':30} {'Units':5}")
    print("-"*40)
    for row in rows:
        no, name, units = row
        print(f"{no:3} {name:30} {units:5}")
    print()
```

The data of students in a group is printed by the following code:

```
def print_group(rows):
    print(f"{'No':3} {'Name':15} {'Mark':4}
            {Grade:5}")
    print("-"*30)
    for row in rows:
        no, name, mark, grade = row
        print(f"{no:3} {name:15} {mark:4}
                {grade:5}")
    print()
```

7. Searching Data:

The table (*student*) is searched by (*no*) by the following code:

```
def get_student(no):
    qry = """SELECT *
             FROM student
             WHERE no = ?"""
    row = con.execute(qry, (no,))
    return row
```

The table (*course*) is searched by (*no*) by the following code:

```
def get_course(no):
    qry = """SELECT *
             FROM course
             WHERE no = ?"""
    row = con.execute(qry, (no,))
    return row
```

The table (*enroll*) is searched by (*sno*) and (*cno*) by the following code:

```
def get_enroll(sno, cno):
    qry = """SELECT *
             FROM enroll
             WHERE sno = ? AND cno = ?"""
    row = con.execute(qry, (sno, cno))
    return row
```

8. Ordering Data:

The students are ordered by (*name*) by the following code:

```
def order_students():
    qry = """SELECT *
             FROM student
             ORDER BY name"""
    rows = con.execute(qry)
    return rows
```

The courses are ordered by (*no*) by the following code:

```
def order_courses():
    qry = """SELECT *
             FROM course
             ORDER BY no"""
    rows = con.execute(qry, (no,))
    return rows
```

The students in a group are ordered by (*name*) by the following code:

```
def order_group(cno):
    qry = """SELECT e.sno, s.name, e.mark,
                    e.grade
             FROM enroll e
             JOIN student s ON e.sno = s.no
             WHERE cno = ?
             ORDER BY s.name"""
    rows = con.execute(qry, (cno,))
    return rows
```

9. Updating Data:

The marks of students in a group are entered by the user, one by one, and updated in the table (*enroll*) by the following code:

```
def update_marks(cno):
    rows = order_group(cno)
    print(f"Enter marks of students ...")
    print(f"{'No':3} {'Name':15} {'Mark':4}")
    print("-"*24)
    i = 0
    for row in rows:
        i += 1
        sno = row[0]
        name = row[1]
        print(f"{i}. {sno:3} {name:15} ",
                end="")
        mark = int(input("? "))
        qry = """UPDATE enroll
                 SET mark = ?
                 WHERE sno = ? AND cno = cno)"""
        con.execute(qry,(mark, sno, cno))
        con.commit()
```

10. Computing Data:

The grades of students in a group are computed and updated in the table (*enroll*) by the following code:

```
def compute_grades(cno):
    rows = order_group(cno)
    for row in rows:
        sno = row[0]
        mark = row[2]
        grade = get_grade(mark)
        qry = """UPDATE enroll
                 SET grade = ?
                 WHERE sno = ? AND cno = cno)"""
        con.execute(qry, (grade, sno, cno))
        con.commit()
```

The function (*get_grade*) is shown by the following code:

```
def get_grade(mark):
    if (mark >= 90):
        grade = "A"
    elif (mark >= 80):
        grade = "B"
    elif (mark >= 70):
        grade = "C"
    elif (mark >= 60):
        grade = "D"
    else:
        grade = "F"
    return grade
```

11. Aggregating Data:

The group statistics are computed using the aggregation functions (COUNT, AVG, MIN, and MAX) and printed by the following code:

```
def group_stats(cno):
    qry = """SELECT COUNT(*), AVG(mark),
                    MIN(mark), MAX(mark)
             FROM enroll
             WHERE cno = ?"""
    row = con.execute(qry, (cno,))
    count, avg, min, max = row
    print(f"Count = {count}")
    print(f"  Avg = {avg:.2f}")
    print(f"  Min = {min}")
    print(f"  Max = {max}")
```

12. Grouping Data:

The grades of students in a group are counted and printed by the following code:

```
def grade_count(cno):
    qry = """SELECT grade, COUNT(grade)
             FROM enroll
             WHERE cno = ?
             GROUP BY grade"""
    rows = con.execute(qry, (cno,))
    print(f"Grade Count")
    print("-"*11)
    for row in rows:
        grade, count = row
        print(f"{grade:5} {count})
```

13. Deleting Data:

The student is deleted from the table (*student*) by (*no*) by the following code:

```
def delete_student(no):
    qry = """DELETE FROM student
             WHERE no = ?"""
    con.execute(qry, (no,))
    con.commit()
```

The course is deleted from the table (*course*) by (*no*) by the following code:

```
def delete_course(no):
    qry = """DELETE FROM course
             WHERE no = ?"""
    con.execute(qry, (no,))
    con.commit()
```

The enrollment is deleted from the table (*enroll*) by (*sno*) and (*cno*) by the following code:

```
def delete_enroll(sno, cno):
    qry = """DELETE FROM enroll
             WHERE sno = ? AND cno = ?"""
    con.execute(qry, (sno, cno))
    con.commit()
```

14. Dropping Tables:

The table (*student*) is deleted from the database by the following code:

```
def drop_student():
    con.execute("DROP TABLE student")
    con.commit()
```

The table (*course*) is deleted from the database by the following code:

```
def drop_course():
    con.execute("DROP TABLE course")
    con.commit()
```

The table (*enroll*) is deleted from the database by the following code:

```
def drop_enroll():
    con.execute("DROP TABLE enroll")
    con.commit()
```

4. RESULTS AND DISCUSSION

The developed program was tested on an experimental data. The program has successfully performed the basic operations of database management using SQLite and provided the required results. The program output is explained step by step in the following section.

Importing Library:

The SQLite library (*sqlite3*) is imported as shown in the following view:

```
import sqlite3
```

Creating Database:

The database (*college.db*) is created and the connection (*con*) is established as shown in the following view:

```
con = create_database("college.db")

Ok: Database 'college' is created
```

Creating Tables:

The table (*student*) is created as shown in the following view:

```
create_student()
```

```
Ok: Table 'student' is created
```

The table (*course*) is created as shown in the following view:

```
create_course()

Ok: Table 'course' is created
```

The table (*enroll*) is created as shown in the following view:

```
create_enroll()

Ok: Table 'enroll' is created
```

Adding Data:

The students are added to the table (*student*) by the user, one by one, as shown in the following view:

```
add_student()

Enter student-no or press <Enter> to quit ...

   No: 101
 Name: Adam
Major: Computer

   No: 102
 Name: Sally
Major: Computer

   No: 103
 Name: John
Major: Computer

   No: 104
 Name: Mary
Major: Computer

   No: 105
 Name: Peter
Major: Computer

   No:

Ok: Students are added
```

The courses are added to the table (*course*) by the user, one by one, as shown in the following view:

```
add_course()

Enter course-no or press <Enter> to quit ...

   No: 100
 Name: Introduction to Computer
Units: 4

   No: 200
 Name: Python Programming
Units: 5

   No: 300
 Name: Data Structures
Units: 5

   No: 400
 Name: Database Systems
Units: 3

   No: 500
 Name: Operating Systems
Units: 3

   No:

Ok: Courses are added
```

The students in a group are added to the table (*enroll*) by the user, one by one, as shown in the following view:

```
add_group(200)

Enter student-no or press <Enter> to quit ...

1. Student No: 101
2. Student No: 102
3. Student No: 103
4. Student No: 104
5. Student No: 105
6. Student No:

Ok: Students are added in course 200
```

Ordering Data:

The students are ordered as shown in the following view:

```
rows = order_students()
```

The courses are ordered as shown in the following view:

```
rows = order_courses()
```

The students in a group are ordered as shown in the following view:

```
rows = order_group(200)
```

Printing Data:

The students are printed as shown in the following view:

```
print_student(rows)

   No  Name            Major           GPA
-------------------------------------------
1. 101 Adam            Computer        0.00
2. 103 John            Computer        0.00
3. 104 Mary            Computer        0.00
4. 105 Peter           Computer        0.00
5. 102 Sally           Computer        0.00
```

The courses are printed as shown in the following view:

```
print_course(rows)

   No  Name                           Units
-----------------------------------------
1. 100 Introduction to Computer       4
2. 200 Python Programming             5
3. 300 Data Structures                5
4. 400 Database Systems               3
5. 500 Operating Systems              3
```

The students in a group are printed as shown in the following view:

```
print_group(rows)

   No  Name            Mark Grade
--------------------------------
1. 101 Adam            0
2. 103 John            0
3. 104 Mary            0
4. 105 Peter           0
```

```
5. 102 Sally           0
```

Updating Data:

The marks of students in a group are entered by the user, one by one, as shown in the following view:

```
update_marks(200)

Enter marks of students ...

   No Name             Mark
----------------------------
1. 101 Adam            ? 90
2. 103 John            ? 80
3. 104 Mary            ? 70
4. 105 Peter           ? 60
5. 102 Sally           ? 50

Ok: Marks are updated
```

Computing Data:

The grades of students in a group are computed and printed as shown in the following view:

```
compute_grades(200)

Ok: Grades are computed

rows = order_group(200)
print_group(rows)

   No  Name             Mark Grade
---------------------------------
1. 101 Adam              90   A
2. 103 John              80   B
3. 104 Mary              70   C
4. 105 Peter             60   D
5. 102 Sally             50   F
```

Aggregating Data:

The group statistics are computed using the aggregation functions (COUNT, AVG, MIN, and MAX) and printed as shown in the following view:

```
group_stats(200)

Count = 5
  Avg = 70.00
  Min = 50
  Max = 90
```

Grouping Data:

The grades of students in a group are counted and printed as shown in the following view:

```
grade_count(200)

Grade Count
-----------
A     1
B     1
C     1
D     1
F     1
```

Deleting Data:

The student is deleted from the table (*student*) as shown in the following view:

```
delete_student(105)

Ok: Student 105 is deleted
```

The course is deleted from the table (*course*) as shown in the following view:

```
delete_course(500)

Ok: Course 500 is deleted
```

The enrollment is deleted from the table (*enroll*) as shown in the following view:

```
delete_enroll(101,200)

Ok: Student 101 is deleted from course 200
```

Dropping Tables:

The table (*student*) is deleted from the database as shown in the following view.

```
drop_student()

Ok: Table 'student' is dropped
```

The table (*course*) is deleted from the database as shown in the following view.

```
drop_course()

Ok: Table 'course' is dropped
```

The table (*enroll*) is deleted from the database as shown in the following view.

```
drop_enroll()

Ok: Table 'enroll' is dropped
```

Now, it is obviously clear that the program has successfully performed the basic operations of database management using SQLite and provided the required results.

5. CONCLUSION

In this research, the goal was to implement database management using SQLite in Python. The literature was reviewed to explore the fundamental concepts of database management using SQLite: database management, database, database operations, structured query language (SQL), SQLite, and Python.

The author developed a program in Python to perform the basic operations of database management using SQLite: importing library, creating database, creating tables, adding data, retrieving data, printing data, searching data, ordering data, updating data, computing data, aggregating data, grouping data, deleting data, and dropping tables.

The developed program was tested on an experimental data. The program has successfully performed the basic operations of database management using SQLite and provided the required results.

In the future, more work is needed to improve the implementation of database management using SQLite. For example, in the development of web and mobile applications.

6. REFERENCES

[1] Sammut, C. & Webb, G. (2011). "Encyclopedia of Machine Learning". Springer.

[2] Kubat, M. (2021). "An Introduction to Machine Learning". Springer.

[3] Jung, A. (2022). "Machine Learning: The Basics". Springer.

[4] Li, H. (2023). "Machine Learning Methods". Springer.

[5] Forsyth, D. (2019). "Applied Machine Learning". Springer.

[6] Chopra, D. & Khurana, R. (2023). "Introduction to Machine Learning with Python". Bentham Science Publishers.

[7] Müller, A. & Guido, S. (2016). "Introduction to Machine Learning with Python: A Guide for Data Scientists". O'Reilly.

[8] Raschka, S. (2015). "Python Machine Learning". Packt Publishing.

[9] Zollanvari, A. (2023). " Machine Learning with Python". Springer.

[10] Sarkar, D., Bali, R., & Sharma, T. (2018). "Practical Machine Learning with Python". Apress.

[11] Zelle, J. (2017). "Python Programming: An Introduction to Computer Science". Franklin, Beedle & Associates.

[12] Padmanabhan, T. (2016). "Programming with Python". Springer.

[13] Chun, W. (2001). "Core Python Programming". Prentice Hall.

[14] Beazley, D. & Jones, B. (2013). "Python Cookbook: Recipes for Mastering Python 3". O'Reilly.

[15] Lutz, M. (2013). "Learning Python: Powerful Object-Oriented Programming". O'Reilly.

[16] Perkovic, L. (2015). "Introduction to Computing using Python: An Application Development Focus". Wiley.

[17] Downey, A. (2016). "Think Python: How to Think Like a Computer Scientist". O'Reilly.

[18] Date, C. (2006). "An introduction to Database Systems". Pearson.

[19] Elmasri, R. & Navathe, S. (2016). "Fundamentals of Database Systems". Pearson.

[20] Silberschatz, A., Korth, H., & Sudarshan, S. (2020). "Database System Concepts". McGraw-Hill.

[21] Connolly, T. & Begg, C. (2015). "Database Systems: A Practical Approach to Design, Implementation, and Management". Pearson.

[22] Ramakrishnan, R. & Gehrke, J. (2003). "Database Management Systems". McGraw-Hill.

[23] Gracia-Molina, H., Ullman, J., & Widom, J. (2009). "Database Systems: The Complete Book". Pearson.

[24] Coronel, C. & Morris, S. (2022). "Database Systems: Design, Implementation, and Management". Cengage Learning.

[25] Churcher, C. (2012). "Beginning Database Design: From Novice to Professional". Apress.

[26] Groff, J., Weinberg, P., & Oppel, A. (2010). "SQL: The Complete Reference". McGraw-Hill.

[27] Batra, R. (2018). "SQL Primer: An Accelerated Introduction to SQL Basics". Apress.

[28] Patrick, J. (2009). " SQL Fundamentals". Pearson.

[29] Nield, T. (2016). "Getting Started with SQL". O'Reilly.

[30] Simon, M. (2023). "Getting Started with SQL and Databases". Springer.

[31] Wilton, P. & Colby, J. (2005). "Beginning SQL". Wiley.

[32] Bush, J. (2020). "Learn SQL Database Programming". Packt Publishing.

[33] Churcher, C. (2016). "Beginning SQL Queries: From Novice to Professional". Apress.

[34] Allen, G. & Owens, M. (2010). "The Definitive Guide to SQLite". Apress.

[35] Kreibich, J. (2010). "Using SQLite". O'Reilly.

[36] Codd, E. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM. 13 (6), 377–87.

[37] Chamberlin, D. & Boyce, R. (1974). "SEQUEL: A Structured English Query Language". Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control, 249–64.

[38] SQLite: http://www.sqlite.org

[39] Python: http://www.python.org

[40] Numpy: http://www.numpy.org

[41] Pandas: http://pandas.pydata.org

[42] Matplotlib: http://www. matplotlib.org

[43] Seaborn: http://seaborn.pydata.org

[44] SciPy: http://scipy.org

[45] NLTK: http://www.nltk.org

[46] SK Learn: http://scikit-learn.org