




submission

-  My Files
-  My Files
-  University

Document Details

Submission ID**trn:oid:::28592:82458400****Submission Date****Feb 17, 2025, 10:56 PM GMT+5:30****Download Date****Feb 17, 2025, 10:56 PM GMT+5:30****File Name****765658 (1).docx****File Size****40.7 KB****20 Pages****2,427 Words****16,553 Characters**





43% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.




Filtered from the Report

- Bibliography
- Quoted Text

Match Groups

-  **58 Not Cited or Quoted 43%**
Matches with neither in-text citation nor quotation marks
-  **0 Missing Quotations 0%**
Matches that are still very similar to source material
-  **0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 29%  Internet sources
- 15%  Publications
- 43%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- 58** Not Cited or Quoted 43%
Matches with neither in-text citation nor quotation marks
- 0** Missing Quotations 0%
Matches that are still very similar to source material
- 0** Missing Citation 0%
Matches that have quotation marks, but no in-text citation
- 0** Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

- 29% Internet sources
- 15% Publications
- 43% Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Submitted works	University of Maryland, Global Campus on 2024-06-26	16%
2	Submitted works	University of Maryland, Global Campus on 2024-02-18	10%
3	Submitted works	University of Maryland, Global Campus on 2024-03-06	6%
4	Internet	www.geeksforgeeks.org	4%
5	Submitted works	University of Maryland, Global Campus on 2023-09-27	2%
6	Publication	Jesper Wisborg Krogh. "MySQL Connector/Python Revealed", Springer Science an...	2%
7	Submitted works	University of Maryland, Global Campus on 2023-09-27	1%
8	Submitted works	University of Maryland, Global Campus on 2024-06-25	<1%
9	Internet	www.coursehero.com	<1%
10	Submitted works	University of Maryland, Global Campus on 2024-02-21	<1%

11

Submitted works

University of Maryland, Global Campus on 2024-09-25

<1%

11

SQL and Python for Database Development

9

Student's Name

Institutional Affiliation

Professor's Name

Course Name

Submission Date

SQL and Python for Database Development

Write syntaxes used for the following actions:

1. **Create** a Database named **mydatabse**.

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword"
)
cursor = conn.cursor()
cursor.execute("CREATE DATABASE mydatabse")
conn.close()
```

2. **Create** a Table within mydatabse named **Customers**

```
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword",
    database="mydatabse"
)
cursor = conn.cursor()
cursor.execute("""
    CREATE TABLE Customers (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255),
        address VARCHAR(255)
    )
""")
conn.close()
```

3. **Insert** data in the **Customers** table.

```
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("James Kerlyson", "123 Elm St"))
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Mary Scott", "456 Oak St"))
conn.commit()
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Bob Kelvin", "456 Oak St"))
```

```
conn.commit()
```

1 4. **Select** all records from the **Customers** table

```
cursor.execute("SELECT * FROM Customers")
for row in cursor.fetchall():
    print(row)
```

5. **Sort** the results of the **Customer** table in **Ascending** and **Descending** order

Ascending order

```
cursor.execute("SELECT * FROM Customers ORDER BY name ASC")
for row in cursor.fetchall():
    print(row)
```

Descending order

```
cursor.execute("SELECT * FROM Customers ORDER BY name DESC")
for row in cursor.fetchall():
    print(row)
```

6. **Delete** a record from the **Customer** table

```
cursor.execute("DELETE FROM Customers WHERE name = 'Mary Scott'")
conn.commit()
```

7. **Update** existing records in the **Customer** table

```
cursor.execute("UPDATE Customers SET address = '789 Maple St' WHERE name = Bob Kelvin")
conn.commit()
```

8. **Join** one or more tables.

```
cursor.execute("""
CREATE TABLE Orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    product VARCHAR(255),
    FOREIGN KEY (customer_id) REFERENCES Customers(id)
)
""")
```

```

1 cursor.execute("""
    SELECT Customers.name, Orders.product
    FROM Customers
    INNER JOIN Orders ON Customers.id = Orders.customer_id
    """)
for row in cursor.fetchall():
    print(row)

```

9. Query Customer database for the selected records

```

6 cursor.execute("SELECT * FROM Customers WHERE address LIKE '%Oak%'")
for row in cursor.fetchall():
    print(row)

```

10. Delete the Customer Table

```

3 cursor.execute("DROP TABLE Customers")
conn.commit()

```

11. Compare Python with SQL for creating databases with respect to ease of learning and application. Which one do you prefer? Why?

Feature	SQL (Direct Querying)	Python (MySQL Connector)
Application	Used mainly for direct database manipulation	Can integrate with applications, automation, and scripts
Flexibility	Fixed syntax for querying	Allows dynamic queries and automation
Ease of Learning	Easier if only SQL is needed	Requires knowledge of Python & SQL
Security	Requires manual security implementations	Can use prepared statements to prevent SQL injection

When working on database management alone, I would prefer SQL due to its direct querying feature.

When integrating several applications together, I would prefer Python due to its flexibility and automation.

Write syntaxes used for the following actions:

12. **Create** a Database named **mydatabse**.

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword"
)
cursor = conn.cursor()
cursor.execute("CREATE DATABASE mydatabse")
conn.close()
```

13. **Create** a Table within mydatabse named **Customers**

```
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword",
    database="mydatabse"
)
cursor = conn.cursor()
cursor.execute("""
    CREATE TABLE Customers (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255),
        address VARCHAR(255)
    )
""")
conn.close()
```

14. **Insert** data in the **Customers** table.

```
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("James Kerlyson", "123 Elm St"))
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Mary Scott", "456 Oak St"))
conn.commit()
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Bob Kelvin", "456 Oak St"))
conn.commit()
```

15. **Select** all records from the **Customers** table

```
cursor.execute("SELECT * FROM Customers")
for row in cursor.fetchall():
    print(row)
```

16. **Sort** the results of the **Customer** table in **Ascending** and **Descending** order

Ascending order

```
cursor.execute("SELECT * FROM Customers ORDER BY name ASC")
for row in cursor.fetchall():
    print(row)
```

Descending order

```
cursor.execute("SELECT * FROM Customers ORDER BY name DESC")
for row in cursor.fetchall():
    print(row)
```

17. **Delete** a record from the **Customer** table

```
cursor.execute("DELETE FROM Customers WHERE name = 'Mary Scott'")
conn.commit()
```

18. **Update** existing records in the **Customer** table

```
cursor.execute("UPDATE Customers SET address = '789 Maple St' WHERE name = Bob Kelvin")
conn.commit()
```

19. **Join** one or more tables.

```
cursor.execute("""
CREATE TABLE Orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    product VARCHAR(255),
    FOREIGN KEY (customer_id) REFERENCES Customers(id)
)
""")

cursor.execute("""
SELECT Customers.name, Orders.product
```

```

FROM Customers
INNER JOIN Orders ON Customers.id = Orders.customer_id
"""
for row in cursor.fetchall():
    print(row)

```

20. Query Customer database for the selected records

```

cursor.execute("SELECT * FROM Customers WHERE address LIKE '%Oak%'")
for row in cursor.fetchall():
    print(row)

```

21. Delete the Customer Table

```

cursor.execute("DROP TABLE Customers")
conn.commit()

```

22. Compare Python with SQL for creating databases with respect to ease of learning and application. Which one do you prefer? Why?

Feature	SQL (Direct Querying)	Python (MySQL Connector)
Application	Used mainly for direct database manipulation	Can integrate with applications, automation, and scripts
Flexibility	Fixed syntax for querying	Allows dynamic queries and automation
Ease of Learning	Easier if only SQL is needed	Requires knowledge of Python & SQL
Security	Requires manual security implementations	Can use prepared statements to prevent SQL injection

When working on database management alone, I would prefer SQL due to its direct querying feature.

When integrating several applications together, I would prefer Python due to its flexibility and automation.

Write syntaxes used for the following actions:

23. Create a Database named mydatabase.

```

import mysql.connector

```

```
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword"
)
cursor = conn.cursor()
cursor.execute("CREATE DATABASE mydatabase")
conn.close()
```

24. **Create** a Table within mydatabse named **Customers**

```
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword",
    database="mydatabase"
)
cursor = conn.cursor()
cursor.execute("""
    CREATE TABLE Customers (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255),
        address VARCHAR(255)
    )
""")
conn.close()
```

25. **Insert** data in the **Customers** table.

```
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("James
Kerlyson", "123 Elm St"))
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Mary
Scott", "456 Oak St"))
conn.commit()
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Bob
Kelvin", "456 Oak St"))
conn.commit()
```

26. **Select** all records from the **Customers** table

```
cursor.execute("SELECT * FROM Customers")
for row in cursor.fetchall():
```

```
print(row)
```

27. **Sort** the results of the **Customer** table in **Ascending** and **Descending** order

Ascending order

```
cursor.execute("SELECT * FROM Customers ORDER BY name ASC")
for row in cursor.fetchall():
    print(row)
```

Descending order

```
cursor.execute("SELECT * FROM Customers ORDER BY name DESC")
for row in cursor.fetchall():
    print(row)
```

28. **Delete** a record from the **Customer** table

```
cursor.execute("DELETE FROM Customers WHERE name = 'Mary Scott'")
conn.commit()
```

29. **Update** existing records in the **Customer** table

```
cursor.execute("UPDATE Customers SET address = '789 Maple St' WHERE name = Bob
Kelvin")
conn.commit()
```

30. **Join** one or more tables.

```
cursor.execute("""
CREATE TABLE Orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    product VARCHAR(255),
    FOREIGN KEY (customer_id) REFERENCES Customers(id)
)
""")
```

```
cursor.execute("""
SELECT Customers.name, Orders.product
FROM Customers
INNER JOIN Orders ON Customers.id = Orders.customer_id
""")
for row in cursor.fetchall():
    print(row)
```

31. Query Customer database for the selected records

```
cursor.execute("SELECT * FROM Customers WHERE address LIKE '%Oak%'")
for row in cursor.fetchall():
    print(row)
```

32. Delete the Customer Table

```
cursor.execute("DROP TABLE Customers")
conn.commit()
```

33. Compare Python with SQL for creating databases with respect to ease of learning and application. Which one do you prefer? Why?

Feature	SQL (Direct Querying)	Python (MySQL Connector)
Application	Used mainly for direct database manipulation	Can integrate with applications, automation, and scripts
Flexibility	Fixed syntax for querying	Allows dynamic queries and automation
Ease of Learning	Easier if only SQL is needed	Requires knowledge of Python & SQL
Security	Requires manual security implementations	Can use prepared statements to prevent SQL injection

When working on database management alone, I would prefer SQL due to its direct querying feature.

When integrating several applications together, I would prefer Python due to its flexibility and automation.

Write syntaxes used for the following actions:

34. Create a Database named mydatabse.

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword"
)
```

```

cursor = conn.cursor()
cursor.execute("CREATE DATABASE mydatabase")
conn.close()

```

35. **Create** a Table within mydatabase named **Customers**

```

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword",
    database="mydatabase"
)
cursor = conn.cursor()
cursor.execute("""
    CREATE TABLE Customers (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255),
        address VARCHAR(255)
    )
""")
conn.close()

```

36. **Insert** data in the **Customers** table.

```

cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("James
Kerlyson", "123 Elm St"))
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Mary
Scott", "456 Oak St"))
conn.commit()
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Bob
Kelvin", "456 Oak St"))
conn.commit()

```

37. **Select** all records from the **Customers** table

```

cursor.execute("SELECT * FROM Customers")
for row in cursor.fetchall():
    print(row)

```

38. **Sort** the results of the **Customer** table in **Ascending** and **Descending** order

Ascending order

```
cursor.execute("SELECT * FROM Customers ORDER BY name ASC")
for row in cursor.fetchall():
    print(row)
```

Descending order

```
cursor.execute("SELECT * FROM Customers ORDER BY name DESC")
for row in cursor.fetchall():
    print(row)
```

39. **Delete** a record from the **Customer** table

```
cursor.execute("DELETE FROM Customers WHERE name = 'Mary Scott'")
conn.commit()
```

40. **Update** existing records in the **Customer** table

```
cursor.execute("UPDATE Customers SET address = '789 Maple St' WHERE name = Bob Kelvin")
conn.commit()
```

41. **Join** one or more tables.

```
cursor.execute("""
CREATE TABLE Orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    product VARCHAR(255),
    FOREIGN KEY (customer_id) REFERENCES Customers(id)
)
""")
```

```
cursor.execute("""
SELECT Customers.name, Orders.product
FROM Customers
INNER JOIN Orders ON Customers.id = Orders.customer_id
""")
for row in cursor.fetchall():
    print(row)
```

42. **Query Customer** database for the selected records

```
cursor.execute("SELECT * FROM Customers WHERE address LIKE '%Oak%'")
for row in cursor.fetchall():
```



```
print(row)
```

3. 43. **Delete** the Customer Table

```
cursor.execute("DROP TABLE Customers")
conn.commit()
```

44. Compare Python with SQL for creating databases with respect to ease of learning and application. Which one do you prefer? Why?

Feature	SQL (Direct Querying)	Python (MySQL Connector)
Application	Used mainly for direct database manipulation	Can integrate with applications, automation, and scripts
Flexibility	Fixed syntax for querying	Allows dynamic queries and automation
Ease of Learning	Easier if only SQL is needed	Requires knowledge of Python & SQL
Security	Requires manual security implementations	Can use prepared statements to prevent SQL injection

When working on database management alone, I would prefer SQL due to its direct querying feature.

When integrating several applications together, I would prefer Python due to its flexibility and automation.

Write syntaxes used for the following actions:

2. 45. **Create** a Database named **mydatabse**.

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword"
)
cursor = conn.cursor()
cursor.execute("CREATE DATABASE mydatabse")
conn.close()
```

46. **Create** a Table within mydatabse named **Customers**

```

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword",
    database="mydatabase"
)
cursor = conn.cursor()
cursor.execute("""
    CREATE TABLE Customers (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255),
        address VARCHAR(255)
    )
""")
conn.close()

```

1 47. **Insert** data in the **Customers** table.

```

cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("James
Kerlyson", "123 Elm St"))
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Mary
Scott", "456 Oak St"))
conn.commit()
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Bob
Kelvin", "456 Oak St"))
conn.commit()

```

1 48. **Select** all records from the **Customers** table

```

cursor.execute("SELECT * FROM Customers")
for row in cursor.fetchall():
    print(row)

```

49. **Sort** the results of the **Customer** table in **Ascending** and **Descending** order

Ascending order

```

cursor.execute("SELECT * FROM Customers ORDER BY name ASC")
for row in cursor.fetchall():
    print(row)

```

Descending order

```
cursor.execute("SELECT * FROM Customers ORDER BY name DESC")
for row in cursor.fetchall():
    print(row)
```

50. **Delete** a record from the **Customer** table

```
cursor.execute("DELETE FROM Customers WHERE name = 'Mary Scott'")
conn.commit()
```

51. **Update** existing records in the **Customer** table

```
cursor.execute("UPDATE Customers SET address = '789 Maple St' WHERE name = Bob Kelvin")
conn.commit()
```

52. **Join** one or more tables.

```
cursor.execute("""
CREATE TABLE Orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    product VARCHAR(255),
    FOREIGN KEY (customer_id) REFERENCES Customers(id)
)
""")
```

```
cursor.execute("""
SELECT Customers.name, Orders.product
FROM Customers
INNER JOIN Orders ON Customers.id = Orders.customer_id
""")
for row in cursor.fetchall():
    print(row)
```

53. **Query Customer** database for the selected records

```
cursor.execute("SELECT * FROM Customers WHERE address LIKE '%Oak%'")
for row in cursor.fetchall():
    print(row)
```

54. **Delete** the Customer Table

```
cursor.execute("DROP TABLE Customers")
```

```
conn.commit()
```

55. Compare Python with SQL for creating databases with respect to ease of learning and application. Which one do you prefer? Why?

Feature	SQL (Direct Querying)	Python (MySQL Connector)
Application	Used mainly for direct database manipulation	Can integrate with applications, automation, and scripts
Flexibility	Fixed syntax for querying	Allows dynamic queries and automation
Ease of Learning	Easier if only SQL is needed	Requires knowledge of Python & SQL
Security	Requires manual security implementations	Can use prepared statements to prevent SQL injection

When working on database management alone, I would prefer SQL due to its direct querying feature.

When integrating several applications together, I would prefer Python due to its flexibility and automation.

Write syntaxes used for the following actions:

2 56. **Create** a Database named **mydatabse**.

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword"
)
cursor = conn.cursor()
cursor.execute("CREATE DATABASE mydatabse")
conn.close()
```

57. **Create** a Table within mydatabse named **Customers**

```
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="yourpassword",
```

```

        database="mydatabase"
    )
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE Customers (
            id INT AUTO_INCREMENT PRIMARY KEY,
            name VARCHAR(255),
            address VARCHAR(255)
        )
    """)
    conn.close()

```

1 58. **Insert** data in the **Customers** table.

```

cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("James
Kerlyson", "123 Elm St"))
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Mary
Scott", "456 Oak St"))
conn.commit()
cursor.execute("INSERT INTO Customers (name, address) VALUES (%s, %s)", ("Bob
Kelvin", "456 Oak St"))
conn.commit()

```

1 59. **Select** all records from the **Customers** table

```

cursor.execute("SELECT * FROM Customers")
for row in cursor.fetchall():
    print(row)

```

60. **Sort** the results of the **Customer** table in **Ascending** and **Descending** order

Ascending order

```

cursor.execute("SELECT * FROM Customers ORDER BY name ASC")
for row in cursor.fetchall():
    print(row)

```

Descending order

```

cursor.execute("SELECT * FROM Customers ORDER BY name DESC")
for row in cursor.fetchall():
    print(row)

```

61. **Delete** a record from the **Customer** table

```

5 cursor.execute("DELETE FROM Customers WHERE name = 'Mary Scott'")
conn.commit()

```

62. Update existing records in the Customer table

```

cursor.execute("UPDATE Customers SET address = '789 Maple St' WHERE name = Bob
Kelvin")
conn.commit()

```

63. Join one or more tables.

```

4 cursor.execute("""
CREATE TABLE Orders (
order_id INT AUTO_INCREMENT PRIMARY KEY,
customer_id INT,
product VARCHAR(255),
FOREIGN KEY (customer_id) REFERENCES Customers(id)
)
""")

```

```

1 cursor.execute("""
SELECT Customers.name, Orders.product
FROM Customers
INNER JOIN Orders ON Customers.id = Orders.customer_id
""")
for row in cursor.fetchall():
print(row)

```

64. Query Customer database for the selected records

```

6 cursor.execute("SELECT * FROM Customers WHERE address LIKE '%Oak%'")
for row in cursor.fetchall():
print(row)

```

65. Delete the Customer Table

```

cursor.execute("DROP TABLE Customers")
conn.commit()

```

66. Compare Python with SQL for creating databases with respect to ease of learning and application. Which one do you prefer? Why?

Feature	SQL (Direct Querying)	Python (MySQL Connector)
Application	Used mainly for direct database manipulation	Can integrate with applications, automation, and scripts
Flexibility	Fixed syntax for querying	Allows dynamic queries and automation
Ease of Learning	Easier if only SQL is needed	Requires knowledge of Python & SQL
Security	Requires manual security implementations	Can use prepared statements to prevent SQL injection

When working on database management alone, I would prefer SQL due to its direct querying feature.

When integrating several applications together, I would prefer Python due to its flexibility and automation.

References

https://www.w3schools.com/python/python_intro.asp

<https://learnpython101.com/database-programming>

<https://www.geeksforgeeks.org/python-database-tutorial/?ref=lbp>

<https://www.opensourceforu.com/2019/04/database-programming-python/>