SN-81F187993CD3

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, 10:57 PM GMT+5:30
File Name	765658 (1).docx
File Size	40.7 KB

20 Pages

2,427 Words

16,553 Characters

0% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

1 AI-generated only 0%
Likely AI-generated text from a large-language model.

2 AI-generated text that was AI-paraphrased 0%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

SQL and Python for Database Development

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 mydatabase")
conn.close()
```

2. ***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()
```

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()
```

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)
    )
""")
```

```
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

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

10. ***Delete*** the Customer Table

```
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 mydatabase")
conn.close()
```

13. ***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()
```

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 **mydatabse**.

```
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 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()
```

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)
```

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:

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 mydatabase")
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()
```

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()
```

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:

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 mydatabase")
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()
```

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()
```

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

```
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.

```
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)
```

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

```
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/