Excellent 👍 — this is an important concept to understand clearly.

Let’s compare the three methods: **.first()**, **.one()**, and **.one\_or\_none()** used in SQLAlchemy queries.

**🧩 1️⃣ .first()**

**Meaning:**  
Return **the first matching record**, or **None** if no result is found.

**Usage:**

student = session.query(Student).filter\_by(name="Priya").first()

**Behavior:**

* ✅ Returns **one object** (the first one found).
* 🚫 Doesn’t care if there are multiple matches.
* ✅ Returns None if no match found.

**Example:**  
If DB has 3 Priyas → you still get just the **first** one.

**🧩 2️⃣ .one()**

**Meaning:**  
Expect **exactly one** matching record.

**Usage:**

student = session.query(Student).filter\_by(name="Priya").one()

**Behavior:**

* ✅ Returns the **object** if **exactly one** row matches.
* ❌ Raises an **error** if:
  + No rows found → NoResultFound
  + More than one row found → MultipleResultsFound

**Use it when:**  
You know your condition (e.g., id=5) will match **exactly one record**.

**🧩 3️⃣ .one\_or\_none()**

**Meaning:**  
Return **one record** if found, or **None** if not found —  
but raise an error if there are **multiple matches**.

**Usage:**

student = session.query(Student).filter\_by(name="Priya").one\_or\_none()

**Behavior:**

* ✅ Returns one object → if **exactly one** match.
* ✅ Returns None → if **no match**.
* ❌ Raises MultipleResultsFound → if **more than one** match.

**🧠 Quick Summary Table:**

| **Method** | **If 1 match** | **If 0 matches** | **If >1 matches** | **Returns** |
| --- | --- | --- | --- | --- |
| .first() | ✅ Returns object | ✅ None | ✅ First object only | Object or None |
| .one() | ✅ Returns object | ❌ Error (NoResultFound) | ❌ Error (MultipleResultsFound) | Object |
| .one\_or\_none() | ✅ Returns object | ✅ None | ❌ Error (MultipleResultsFound) | Object or None |

**💡 In simple words:**

* **.first()** → “Give me *something* if it exists.”
* **.one()** → “There *must* be exactly one — no more, no less.”
* **.one\_or\_none()** → “There can be *zero or one*, but not more.”

Perfect 👍 — let’s build a **small, clear SQLAlchemy + MySQL** example to show how  
.first(), .one(), and .one\_or\_none() behave differently.

**🧱 Step-by-step Example**

**1️⃣ Import and Setup**

from sqlalchemy import create\_engine, Column, Integer, String

from sqlalchemy.orm import declarative\_base, sessionmaker

from sqlalchemy.exc import NoResultFound, MultipleResultsFound

# Create base class for ORM models

Base = declarative\_base()

# MySQL connection URL (replace with your own credentials)

# Example: user = root, password = root, database = testdb

engine = create\_engine("mysql+mysqlconnector://root:root@localhost/testdb")

# Create session

Session = sessionmaker(bind=engine)

session = Session()

**2️⃣ Define a Simple Model (Table)**

class Student(Base):

\_\_tablename\_\_ = "students"

id = Column(Integer, primary\_key=True)

name = Column(String(50))

age = Column(Integer)

Create the table (if it doesn’t exist):

Base.metadata.create\_all(engine)

**3️⃣ Insert Some Sample Data**

# Clear existing data (for demo)

session.query(Student).delete()

# Insert new records

session.add\_all([

Student(name="Priya", age=20),

Student(name="Priya", age=22),

Student(name="Rohan", age=21)

])

session.commit()

✅ Now table students looks like:

| **id** | **name** | **age** |
| --- | --- | --- |
| 1 | Priya | 20 |
| 2 | Priya | 22 |
| 3 | Rohan | 21 |

**4️⃣ Demonstrating .first(), .one(), .one\_or\_none()**

**🔹 .first()**

student = session.query(Student).filter\_by(name="Priya").first()

print("FIRST:", student.name, student.age if student else "Not found")

🟢 Output →

FIRST: Priya 20

👉 Returns the *first* matching row.

**🔹 .one()**

try:

student = session.query(Student).filter\_by(name="Priya").one()

print("ONE:", student.name, student.age)

except NoResultFound:

print("ONE: No result found")

except MultipleResultsFound:

print("ONE: More than one result found!")

🔴 Output →

ONE: More than one result found!

👉 Because two "Priya" rows exist.

**🔹 .one\_or\_none()**

try:

student = session.query(Student).filter\_by(name="Rohan").one\_or\_none()

print("ONE\_OR\_NONE:", student.name, student.age if student else "Not found")

except MultipleResultsFound:

print("ONE\_OR\_NONE: More than one result found!")

🟢 Output →

ONE\_OR\_NONE: Rohan 21

Now test a name that doesn’t exist:

student = session.query(Student).filter\_by(name="Anjali").one\_or\_none()

print("ONE\_OR\_NONE:", student)

🟢 Output →

ONE\_OR\_NONE: None

**🧠 Summary**

| **Method** | **Result (if 2 Priyas)** | **Result (if 0 matches)** | **Safe to use when** |
| --- | --- | --- | --- |
| .first() | Returns first Priya | Returns None | You just need any one match |
| .one() | ❌ Error | ❌ Error | You expect exactly 1 record |
| .one\_or\_none() | ❌ Error | ✅ None | You expect 0 or 1 record |

Would you like me to extend this same script to also **update and delete** using the record fetched by .first() (so your students can see the full CRUD cycle)?

Excellent 👏 — let’s extend the same SQLAlchemy + MySQL example into a **complete CRUD (Create, Read, Update, Delete)** demo.

This will show how .first() is used not just to fetch data but also to **update** and **delete** records easily.

**🧱 Full CRUD Example (using .first() and MySQL)**

**1️⃣ Import and Setup**

from sqlalchemy import create\_engine, Column, Integer, String

from sqlalchemy.orm import declarative\_base, sessionmaker

# Base class for ORM models

Base = declarative\_base()

# MySQL connection URL (replace credentials as needed)

# Format: mysql+mysqlconnector://username:password@host/dbname

engine = create\_engine("mysql+mysqlconnector://root:root@localhost/testdb")

# Create session

Session = sessionmaker(bind=engine)

session = Session()

**2️⃣ Define Model**

class Student(Base):

\_\_tablename\_\_ = "students"

id = Column(Integer, primary\_key=True)

name = Column(String(50))

age = Column(Integer)

def \_\_repr\_\_(self):

return f"<Student(id={self.id}, name='{self.name}', age={self.age})>"

Create the table (if not already there):

Base.metadata.create\_all(engine)

**3️⃣ CREATE (Insert Records)**

# Clear old data for a fresh start

session.query(Student).delete()

# Add students

students = [

Student(name="Priya", age=20),

Student(name="Rohan", age=21),

Student(name="Anjali", age=19)

]

session.add\_all(students)

session.commit()

print("✅ Records inserted successfully!")

**4️⃣ READ (Query Records)**

Fetch all records:

print("\nAll Students:")

for s in session.query(Student).all():

print(s)

Fetch one record using .first():

student = session.query(Student).filter\_by(name="Priya").first()

print("\nFetched with .first():", student)

**5️⃣ UPDATE (Modify a Record)**

Let’s update **Priya’s age** from 20 → 22:

student\_to\_update = session.query(Student).filter\_by(name="Priya").first()

if student\_to\_update:

print("\nBefore Update:", student\_to\_update)

student\_to\_update.age = 22 # Change field value

session.commit()

print("After Update:", student\_to\_update)

else:

print("No student found with that name.")

✅ Output:

Before Update: <Student(id=1, name='Priya', age=20)>

After Update: <Student(id=1, name='Priya', age=22)>

**6️⃣ DELETE (Remove a Record)**

Let’s delete **Anjali**:

student\_to\_delete = session.query(Student).filter\_by(name="Anjali").first()

if student\_to\_delete:

session.delete(student\_to\_delete)

session.commit()

print("\nDeleted:", student\_to\_delete)

else:

print("No student found to delete.")

Check remaining records:

print("\nRemaining Students:")

for s in session.query(Student).all():

print(s)

**🧠 Final Output Summary (example)**

✅ Records inserted successfully!

All Students:

<Student(id=1, name='Priya', age=20)>

<Student(id=2, name='Rohan', age=21)>

<Student(id=3, name='Anjali', age=19)>

Fetched with .first(): <Student(id=1, name='Priya', age=20)>

Before Update: <Student(id=1, name='Priya', age=20)>

After Update: <Student(id=1, name='Priya', age=22)>

Deleted: <Student(id=3, name='Anjali', age=19)>

Remaining Students:

<Student(id=1, name='Priya', age=22)>

<Student(id=2, name='Rohan', age=21)>

**💡 Key Learning:**

* .first() → great for **fetching one record** before **updating or deleting**.
* Always **commit()** after modifying or deleting.
* Using the ORM feels like working with normal Python objects,  
  but behind the scenes, SQLAlchemy converts them to SQL queries.

Would you like me to extend this with a **.one\_or\_none() based safe update** (so that update happens only if exactly one record matches)? That’s a great next teaching step for students.

Perfect 👏 — now let’s extend the same CRUD example to use **.one\_or\_none()** for **safe updating**, where the update happens **only if exactly one record matches**.

This helps prevent accidental updates when multiple rows share the same name.

**🧠 Why use .one\_or\_none() for Update?**

Sometimes multiple records can have the same value (like "Priya").  
If you use .first(), it will just update the first one — but that might not be what you intended.

Using .one\_or\_none() ensures:

* ✅ Update only if **exactly one** record matches.
* ⚠️ If no match → do nothing.
* ❌ If more than one match → raise an error (MultipleResultsFound).

**🧱 Safe Update Example (with .one\_or\_none())**

**Code Continuation (after your CRUD example)**

from sqlalchemy.exc import MultipleResultsFound

# Safe Update using one\_or\_none()

try:

student\_to\_update = session.query(Student).filter\_by(name="Rohan").one\_or\_none()

if student\_to\_update:

print("\nBefore Safe Update:", student\_to\_update)

student\_to\_update.age = 25

session.commit()

print("After Safe Update:", student\_to\_update)

else:

print("No student found with that name.")

except MultipleResultsFound:

print("❌ Multiple students found with that name! Update cancelled.")

**✅ Output (if exactly one "Rohan" exists)**

Before Safe Update: <Student(id=2, name='Rohan', age=21)>

After Safe Update: <Student(id=2, name='Rohan', age=25)>

**⚠️ Output (if there are multiple "Rohan" records)**

❌ Multiple students found with that name! Update cancelled.

**⚡ Tip for Students:**

| **Method** | **When to Use** | **Behavior** |
| --- | --- | --- |
| .first() | You just want one record (any one) | Returns first or None |
| .one() | You *know* exactly one record must exist | Raises error if 0 or >1 |
| .one\_or\_none() | You expect 0 or 1 record safely | Returns None or record; error if >1 |

**🧩 Optional — Handle by Unique Column**

Usually, in production, you’ll use a **unique identifier** (like id) for updates:

student\_to\_update = session.query(Student).filter\_by(id=2).one\_or\_none()

This ensures no ambiguity — .one\_or\_none() will **always** find at most one record.

Would you like me to now extend this example with a **safe delete** using .one\_or\_none() as well (so students can compare safe update vs safe delete)?

Perfect 👍 — let’s now add the **safe delete** operation using .one\_or\_none() — this is a great way to show students how to safely remove records in SQLAlchemy **without accidentally deleting multiple rows**.

**🧱 Safe DELETE Example (using .one\_or\_none())**

We’ll continue from the same example setup (with Student model, MySQL engine, and session already created).

**🧩 Safe Delete Code**

from sqlalchemy.exc import MultipleResultsFound

# Safe delete using one\_or\_none()

try:

student\_to\_delete = session.query(Student).filter\_by(name="Priya").one\_or\_none()

if student\_to\_delete:

print("\nBefore Delete:", student\_to\_delete)

session.delete(student\_to\_delete)

session.commit()

print("✅ Record deleted safely!")

else:

print("ℹ️ No student found with that name.")

except MultipleResultsFound:

print("❌ Multiple students found with that name! Delete cancelled.")

**✅ Case 1: Only one record with that name**

If only one "Priya" exists →  
**Output:**

Before Delete: <Student(id=1, name='Priya', age=22)>

✅ Record deleted safely!

**⚠️ Case 2: Multiple records with same name**

If there are multiple "Priya" entries →  
**Output:**

❌ Multiple students found with that name! Delete cancelled.

➡️ This prevents accidental deletion of multiple rows with the same name.

**🧠 Why .one\_or\_none() is safer than .first() or .filter\_by().delete()**

| **Method** | **Behavior** | **Risk** |
| --- | --- | --- |
| .filter\_by(name="Priya").delete() | Deletes all matching records at once | ❌ Dangerous (bulk delete) |
| .first() | Deletes only the first match | ⚠️ Unpredictable if duplicates exist |
| .one\_or\_none() | Deletes only if exactly 1 match | ✅ Safe and explicit |

**⚡ Real-world Best Practice:**

Always use a **unique column** (like id or email) for updates and deletes:

student\_to\_delete = session.query(Student).filter\_by(id=2).one\_or\_none()

That way, you never have ambiguity.

**✅ After Safe Delete, Check Remaining Records**

print("\nRemaining Students:")

for s in session.query(Student).all():

print(s)

Would you like me to combine everything (insert, read, update, safe update, delete, safe delete) into **one complete script** that students can run from start to end?

Excellent 👏 — here’s the **complete SQLAlchemy + MySQL CRUD example**, all in **one runnable script**, written in a **clear and educational style** for students.

It demonstrates:

* CREATE (Insert)
* READ (Query)
* UPDATE (with .first() and .one\_or\_none())
* DELETE (with .first() and .one\_or\_none())
* Safe handling of multiple matches

**🧱 Complete Script — SQLAlchemy + MySQL (CRUD with Safe Operations)**

from sqlalchemy import create\_engine, Column, Integer, String

from sqlalchemy.orm import declarative\_base, sessionmaker

from sqlalchemy.exc import NoResultFound, MultipleResultsFound

# ------------------------------------------------------------

# 1️⃣ Define Base and Engine

# ------------------------------------------------------------

Base = declarative\_base()

# MySQL connection (change credentials if needed)

engine = create\_engine("mysql+mysqlconnector://root:root@localhost/testdb")

Session = sessionmaker(bind=engine)

session = Session()

# ------------------------------------------------------------

# 2️⃣ Define ORM Model

# ------------------------------------------------------------

class Student(Base):

\_\_tablename\_\_ = "students"

id = Column(Integer, primary\_key=True)

name = Column(String(50))

age = Column(Integer)

def \_\_repr\_\_(self):

return f"<Student(id={self.id}, name='{self.name}', age={self.age})>"

# Create table if not exists

Base.metadata.create\_all(engine)

# ------------------------------------------------------------

# 3️⃣ CREATE (Insert Data)

# ------------------------------------------------------------

# Clear old data

session.query(Student).delete()

# Insert fresh records

students = [

Student(name="Priya", age=20),

Student(name="Rohan", age=21),

Student(name="Anjali", age=19)

]

session.add\_all(students)

session.commit()

print("\n✅ Records inserted successfully!")

# ------------------------------------------------------------

# 4️⃣ READ (Fetch Data)

# ------------------------------------------------------------

print("\nAll Students:")

for s in session.query(Student).all():

print(s)

# Fetch first record using .first()

student = session.query(Student).filter\_by(name="Priya").first()

print("\nFetched with .first():", student)

# ------------------------------------------------------------

# 5️⃣ UPDATE (Simple Update using .first())

# ------------------------------------------------------------

student\_to\_update = session.query(Student).filter\_by(name="Priya").first()

if student\_to\_update:

print("\nBefore Update:", student\_to\_update)

student\_to\_update.age = 22

session.commit()

print("After Update:", student\_to\_update)

else:

print("No student found to update.")

# ------------------------------------------------------------

# 6️⃣ SAFE UPDATE (Using .one\_or\_none())

# ------------------------------------------------------------

try:

student\_to\_update = session.query(Student).filter\_by(name="Rohan").one\_or\_none()

if student\_to\_update:

print("\nBefore Safe Update:", student\_to\_update)

student\_to\_update.age = 25

session.commit()

print("After Safe Update:", student\_to\_update)

else:

print("ℹ️ No student found with that name.")

except MultipleResultsFound:

print("❌ Multiple students found! Update cancelled.")

# ------------------------------------------------------------

# 7️⃣ DELETE (Simple Delete using .first())

# ------------------------------------------------------------

student\_to\_delete = session.query(Student).filter\_by(name="Anjali").first()

if student\_to\_delete:

print("\nDeleting:", student\_to\_delete)

session.delete(student\_to\_delete)

session.commit()

print("✅ Record deleted successfully!")

else:

print("No student found to delete.")

# ------------------------------------------------------------

# 8️⃣ SAFE DELETE (Using .one\_or\_none())

# ------------------------------------------------------------

try:

student\_to\_delete = session.query(Student).filter\_by(name="Priya").one\_or\_none()

if student\_to\_delete:

print("\nBefore Safe Delete:", student\_to\_delete)

session.delete(student\_to\_delete)

session.commit()

print("✅ Record deleted safely!")

else:

print("ℹ️ No student found with that name.")

except MultipleResultsFound:

print("❌ Multiple students found! Delete cancelled.")

# ------------------------------------------------------------

# 9️⃣ SHOW FINAL DATA

# ------------------------------------------------------------

print("\nRemaining Students in Table:")

for s in session.query(Student).all():

print(s)

# ------------------------------------------------------------

# 🔚 END

# ------------------------------------------------------------

print("\n✅ All CRUD operations completed successfully.")

**🧠 What Students Learn from This**

| **Concept** | **Shown In Code** |
| --- | --- |
| Base class and ORM mapping | Base = declarative\_base() |
| Table creation | Base.metadata.create\_all(engine) |
| Create records | session.add\_all([...]) |
| Read records | .all() and .first() |
| Update | .first() and .one\_or\_none() |
| Delete | .first() and .one\_or\_none() |
| Safe exception handling | MultipleResultsFound, NoResultFound |
| Commit changes | session.commit() |

**💡 Notes for Students:**

* Always **commit()** after any insert, update, or delete.
* Use .one\_or\_none() for **safe operations** when you expect **0 or 1 record**.
* Use unique columns (like id) for updates and deletes whenever possible.
* You can replace filter\_by(name="...") with filter(Student.name == "...") — both work.

Would you like me to make a **version of this same script for SQLite** (so students can run it instantly without needing MySQL setup)?