When we use **raw MySQL (like mysql.connector or pymysql)**, we manually create a **cursor**.  
But when we use **SQLAlchemy ORM**, things change.

**When you use plain MySQL in Python**

Example:

import mysql.connector

conn = mysql.connector.connect(

host="localhost",

user="root",

password="1234",

database="testdb"

)

cursor = conn.cursor() # You create a cursor manually

cursor.execute("SELECT \* FROM products")

rows = cursor.fetchall()

**Here:**

* The **cursor** is your tool to send SQL commands directly (SELECT, INSERT, UPDATE, etc.).
* You have to write SQL statements yourself as strings.
* You manage transactions manually (commit, rollback).

**When you use SQLAlchemy ORM**

Example:

engine = create\_engine("mysql+mysqlconnector://root:1234@localhost/testdb")

Base.metadata.create\_all(engine)

Session = sessionmaker(bind=engine)

session = Session()

**Here’s what happens internally:**

* engine creates a **connection pool** (manages DB connections under the hood).
* When you call session = Session(), SQLAlchemy **automatically obtains a connection** from that pool.
* That connection **internally creates a cursor** when executing queries — **but you never see or manage it directly**.

You just do:

session.add(product)

session.commit()

products = session.query(Product).all()

and **SQLAlchemy ORM**:

1. Converts your Python code (like .add() or .query()) into SQL strings.
2. Creates a **cursor behind the scenes**.
3. Executes that SQL.
4. Maps the SQL results back to Python objects (ORM model instances).

**In short:**

**Session in SQLAlchemy ORM = Smart version of connection + cursor combined.**  
It hides all low-level SQL details and lets you work with Python classes and objects instead.