**In Python (SQLAlchemy ORM)**

SQLAlchemy **supports transactions**, but it gives **you two levels** of control:

1. **Manual (explicit)** — like JDBC or plain transactions
2. **Automatic (context-managed)** — similar to JPA’s automatic transaction handling

**1 Explicit Transaction (Manual Commit)**

This is the common way when you’re not using a framework like Flask or FastAPI:

session = Session()

try:

product = Product(name="Laptop", price=55000)

session.add(product)

session.commit() # Commit explicitly

except:

session.rollback() # Rollback on error

finally:

session.close()

Here, you’re controlling the transaction manually.

**2 Automatic Transaction (Context Manager Style)**

SQLAlchemy lets you do this:

from sqlalchemy.orm import Session

with Session(engine) as session:

with session.begin(): # This starts a transaction automatically

product = Product(name="Phone", price=35000)

session.add(product)

What happens here:

* When session.begin() starts → a transaction opens.
* When the with block exits successfully → **commit happens automatically**.
* If an exception occurs → **rollback** happens automatically.
* You don’t have to manually call commit() or rollback().

SQLAlchemy **does support automatic transactions**, but **only if you use the context-managed pattern** or integrate it with frameworks like:

* Flask SQLAlchemy
* FastAPI with SQLAlchemy Session dependency injection

In those cases, you don’t explicitly call commit() — it’s managed automatically, very much like Spring’s @Transactional.

**Example**

with Session(engine) as session:

with session.begin():

p1 = Product(name="TV", price=40000)

p2 = Product(name="Refrigerator", price=60000)

session.add\_all([p1, p2])

# Auto commit on success, auto rollback on failure