inline-basic_association.py

Link

> https://github.com/sqlalchemy/sqlalchemy/blob/8503dc2e948908199cd8ba4e6b1d1ddcf92f402(

▼ Code

```
class OrderItem(Base):
    __tablename__ = "orderitem"
    order_id = Column(Integer, ForeignKey("order.order_id"), primary_key=True)
    item_id = Column(Integer, ForeignKey("item.item_id"), primary_key=True)
    price = Column(Float, nullable=False)

    def __init__(self, item, price=None):
        self.item = item
        self.price = price or item.price

    item = relationship(Item, lazy="joined")


if __name__ == "__main__":
    engine = create_engine("sqlite://")
    Base.metadata.create_all(engine)

    session = Session(engine)

    tshirt, mug, hat, crowbar = (
        Item("SA T-Shirt", 10.99),
        Item("SA Mug", 6.50),
        Item("SA Hat", 8.99),
        Item("MySQL Crowbar", 16.99),
    )
    session.add_all([tshirt, mug, hat, crowbar])
    session.commit()

    order = Order("john smith")

    order.order_items.append(OrderItem(mug))
    order.order_items.append(OrderItem(crowbar, 10.99))
    order.order_items.append(OrderItem(hat))
    session.add(order)
    session.commit()

    order = session.query(Order).filter_by(customer_name="john smith").one()
    print(
        [
            (order_item.item.description, order_item.price)
            for order_item in order.order_items
        ]
    )
```

```
    q = session.query(Order).join("order_items", "item")
    q = q.filter(
        and_(Item.description == "MySQL Crowbar", Item.price > OrderItem.price)
    )

    print([order.customer_name for order in q])
```

▼ Documentation

```
class OrderItem(Base):
    __tablename__ = "orderitem"
    order_id = Column(Integer, ForeignKey("order.order_id"), primary_key=True)
    item_id = Column(Integer, ForeignKey("item.item_id"), primary_key=True)
    price = Column(Float, nullable=False)

    def __init__(self, item, price=None):
        self.item = item
        self.price = price or item.price

    item = relationship(Item, lazy="joined")


if __name__ == "__main__":
    engine = create_engine("sqlite://")
    Base.metadata.create_all(engine)

    session = Session(engine)

    # create catalog
    tshirt, mug, hat, crowbar = (
        Item("SA T-Shirt", 10.99),
        Item("SA Mug", 6.50),
        Item("SA Hat", 8.99),
        Item("MySQL Crowbar", 16.99),
    )
    session.add_all([tshirt, mug, hat, crowbar])
    session.commit()

    # create an order
    order = Order("john smith")

    # add three OrderItem associations to the Order and save
    order.order_items.append(OrderItem(mug))
    order.order_items.append(OrderItem(crowbar, 10.99))
    order.order_items.append(OrderItem(hat))
    session.add(order)
    session.commit()

    # query the order, print items
    order = session.query(Order).filter_by(customer_name="john smith").one()
    print(
```

```
        [
            (order_item.item.description, order_item.price)
            for order_item in order.order_items
        ]
    )

    # print customers who bought 'MySQL Crowbar' on sale
    q = session.query(Order).join("order_items", "item")
    q = q.filter(
        and_(Item.description == "MySQL Crowbar", Item.price > OrderItem.price)
    )

    print([order.customer_name for order in q
```

Prompt

> As a code documentation assistant, you are assigned to document at the in-line level. When in-line comments are needed, your task is to insert comments within the code itself. The output should be the entire code along with the documentation you've added.
> The code is as follows:
>
> """
> class OrderItem(Base):
> __tablename__ = "orderitem"
> order_id = Column(Integer, ForeignKey("order.order_id"), primary_key=True)
> item_id = Column(Integer, ForeignKey("item.item_id"), primary_key=True)
> price = Column(Float, nullable=False)
>
> def __init__(self, item, price=None):
> self.item = item
> self.price = price or item.price
>
> item = relationship(Item, lazy="joined")
>
>
> if __name__ == "__main__":
> engine = create_engine("sqlite://")
> Base.metadata.create_all(engine)
>
> session = Session(engine)
>
> tshirt, mug, hat, crowbar = (
> Item("SA T-Shirt", 10.99),
> Item("SA Mug", 6.50),

```
        Item("SA Hat", 8.99),
        Item("MySQL Crowbar", 16.99),
    )
    session.add_all([tshirt, mug, hat, crowbar])
    session.commit()

    order = Order("john smith")

    order.order_items.append(OrderItem(mug))
    order.order_items.append(OrderItem(crowbar, 10.99))
    order.order_items.append(OrderItem(hat))
    session.add(order)
    session.commit()

    order = session.query(Order).filter_by(customer_name="john smith").one()
    print(
        [
            (order_item.item.description, order_item.price)
            for order_item in order.order_items
        ]
    )



    q = session.query(Order).join("order_items", "item")
    q = q.filter(
        and_(Item.description == "MySQL Crowbar", Item.price > OrderItem.price)
    )

    print([order.customer_name for order in q])
"""
```

Responses

ChatGPT

```
# Import necessary libraries
from sqlalchemy import Column, Integer, Float, ForeignKey, create_engine, and_
from sqlalchemy.orm import relationship, Session
from sqlalchemy.ext.declarative import declarative_base

# Create a base class for declarative models
Base = declarative_base()

# Define the OrderItem class
class OrderItem(Base):
```

```
    # Define the table name
    __tablename__ = "orderitem"
    # Define columns with their data types and relationships
    order_id = Column(Integer, ForeignKey("order.order_id"), primary_key=True)
    item_id = Column(Integer, ForeignKey("item.item_id"), primary_key=True)
    price = Column(Float, nullable=False)

    # Constructor to initialize an OrderItem instance
    def __init__(self, item, price=None):
        self.item = item
        self.price = price or item.price

    # Define a relationship with the Item class
    item = relationship(Item, lazy="joined")

# Check if the script is run as the main program
if __name__ == "__main__":
    # Create a SQLite database engine
    engine = create_engine("sqlite://")
    # Create tables in the database based on the defined models
    Base.metadata.create_all(engine)

    # Create a session to interact with the database
    session = Session(engine)

    # Create instances of Item objects
    tshirt, mug, hat, crowbar = (
        Item("SA T-Shirt", 10.99),
        Item("SA Mug", 6.50),
        Item("SA Hat", 8.99),
        Item("MySQL Crowbar", 16.99),
    )

    # Add the Item instances to the session and commit changes to the database
    session.add_all([tshirt, mug, hat, crowbar])
    session.commit()

    # Create an Order instance
    order = Order("john smith")

    # Add OrderItem instances to the Order
    order.order_items.append(OrderItem(mug))
    order.order_items.append(OrderItem(crowbar, 10.99))
    order.order_items.append(OrderItem(hat))

    # Add the Order to the session and commit changes
    session.add(order)
    session.commit()

    # Query the database to retrieve the customer's order items
    order = session.query(Order).filter_by(customer_name="john smith").one()
    print(
        [
            (order_item.item.description, order_item.price)
```

```
            for order_item in order.order_items
        ]
    )

    # Query the database to find orders with specific criteria
    q = session.query(Order).join("order_items", "item")
    q = q.filter(
        and_(Item.description == "MySQL Crowbar", Item.price > OrderItem.price)
    )

    # Print the customer names for the matching orders
    print([order.customer_name for order in q])
```

GPT4

```
# Import necessary modules and classes
from sqlalchemy import create_engine, Column, Integer, Float, ForeignKey
from sqlalchemy.orm import relationship, Session
from sqlalchemy.ext.declarative import declarative_base

# Define base model for SQLAlchemy ORM
Base = declarative_base()

# Define the OrderItem class representing items in an order
class OrderItem(Base):
    __tablename__ = "orderitem"  # Set the table name for the OrderItem model

    # Define columns for the table
    order_id = Column(Integer, ForeignKey("order.order_id"), primary_key=True)  # Order ID as a
    item_id = Column(Integer, ForeignKey("item.item_id"), primary_key=True)    # Item ID as a pr
    price = Column(Float, nullable=False)  # Price of the item, must be provided (non-nullable)

    def __init__(self, item, price=None):
        # Constructor for creating an OrderItem instance
        self.item = item  # Set the item for the order item
        self.price = price or item.price  # Set the price, defaulting to the item's price if not

    # Establish a relationship with the Item model for easy data retrieval
    item = relationship(Item, lazy="joined")

# Main execution block
if __name__ == "__main__":
    # Database setup and initialization
    engine = create_engine("sqlite://")  # Create an in-memory SQLite database
    Base.metadata.create_all(engine)  # Create all tables in the database

    # Create a new session for database operations
    session = Session(engine)

    # Create item instances
    tshirt, mug, hat, crowbar = (
        Item("SA T-Shirt", 10.99),
        Item("SA Mug", 6.50),
```

```
        Item("SA Hat", 8.99),
        Item("MySQL Crowbar", 16.99),
    )

    # Add created items to the session and commit to the database
    session.add_all([tshirt, mug, hat, crowbar])
    session.commit()

    # Create an order instance for the customer 'john smith'
    order = Order("john smith")

    # Append items to the order
    order.order_items.append(OrderItem(mug))
    order.order_items.append(OrderItem(crowbar, 10.99))
    order.order_items.append(OrderItem(hat))

    # Add the order to the session and commit to the database
    session.add(order)
    session.commit()

    # Query the database for the order placed by 'john smith' and print details
    order = session.query(Order).filter_by(customer_name="john smith").one()
    print(
        [
            (order_item.item.description, order_item.price)
            for order_item in order.order_items
        ]
    )

    # Advanced query to find orders where the item 'MySQL Crowbar' was sold for less than its li
    q = session.query(Order).join("order_items", "item")
    q = q.filter(
        and_(Item.description == "MySQL Crowbar", Item.price > OrderItem.price)
    )

    # Print the names of customers who bought 'MySQL Crowbar' for less than its listed price
    print([order.customer_name for order in q])
```

Bard

```
Unable to produce a tangible output for the same prompt
```

LLama2

```
# Define a class `OrderItem` that inherits from `Base`
class OrderItem(Base):
    # Set the table name to "orderitem"
    __tablename__ = "orderitem"

    # Define columns:
    #   - order_id (integer, foreign key referencing "order.order_id", primary key)
    #   - item_id (integer, foreign key referencing "item.item_id", primary key)
    #   - price (float, non-nullable)
```

```
    order_id = Column(Integer, ForeignKey("order.order_id"), primary_key=True)
    item_id = Column(Integer, ForeignKey("item.item_id"), primary_key=True)
    price = Column(Float, nullable=False)

    # Initialize instance variables
    def __init__(self, item, price=None):
        # Initialize instance variable 'item'
        self.item = item

        # If 'price' argument is not provided, set it to the item's price
        self.price = price or item.price

    # Define a relationship between OrderItem and Item
    item = relationship(Item, lazy="joined")

# Create an engine and create all tables
if __name__ == "__main__":
    engine = create_engine("sqlite://")
    Base.metadata.create_all(engine)

# Create a session using the engine
session = Session(engine)

# Add some items to the database
tshirt, mug, hat, crowbar = (
    Item("SA T-Shirt", 10.99),
    Item("SA Mug", 6.50),
    Item("SA Hat", 8.99),
    Item("MySQL Crowbar", 16.99),
)
session.add_all([tshirt, mug, hat, crowbar])
session.commit()

# Create an order and add some order items to it
order = Order("john smith")
order.order_items.append(OrderItem(mug))
order.order_items.append(OrderItem(crowbar, 10.99))
order.order_items.append(OrderItem(hat))
session.add(order)
session.commit()

# Query the database to retrieve an order and its items
order = session.query(Order).filter_by(customer_name="john smith").one()
print(
    [
        (order_item.item.description, order_item.price)
        for order_item in order.order_items
    ]
)

# Join orders and items on the order items table
q = session.query(Order).join("order_items", "item")

# Filter the results based on item description and price
```

```
q = q.filter(
    and_(Item.description == "MySQL Crowbar", Item.price > OrderItem.price)
)

# Print the customer names of the matching orders
print([order.customer_name for order in q])
```

StarChat

```
# models.py
from sqlalchemy import Column, Integer, String, Float,ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base
import decimal

Base = declarative_base()

class Item(Base):
    __tablename__ = 'item'

    item_id = Column(Integer,primary_key=True)
    description = Column(String(200),nullable=False)
    price =Column(Float,nullable=False)

    def __repr__(self):
        return f"<Item(item_id={self.item_id},description='{self.description}',price={self.price

class Order(Base):
    __tablename__ ="order"

    order_id = Column(Integer,primary_key=True)
    customer_name = Column(String(80),nullable=False)
    shipping_address = Column(String(200))
    billing_address = Column(String(200))
    total = Column(Float,default=decimal.Decimal('0'),nullable=False)

    def __repr__(self):
        return f"<Order(order_id={self.order_id},customer_name='{self.customer_name}')>"


class OrderItem(Base):
    __tablename__ = "orderitem"
    order_id = Column(Integer, ForeignKey("order.order_id"), primary_key=True)
    item_id = Column(Integer, ForeignKey("item.item_id"), primary_key=True)
    price = Column(Float, nullable=False)

    def __init__(self, item, price=None):
        self.item = item
        self.price = price or item.price

    item = relationship(Item, lazy="joined")
```

```
# main.py
from models import *
from sqlalchemy import create_engine
from sqlalchemy.orm importSession

engine = create_engine("sqlite:///catalog.db")
Base.metadata.create_all(engine)
session = Session(bind=engine)

# Create some test data
tshirt, mug, hat, crowbar = (
    Item("SA T-Shirt", 10.99),
    Item("SA Mug", 6.50),
    Item("SA Hat", 8.99),
    Item("MySQL```
```