

file:basic_association.py

Link

[sqlalchemy/examples/association/basic_association.py at 8503dc2e948908199cd8ba4e6b1d1ddcf92f4020 · sqlalchemy/sqlalchemy · GitHub](https://github.com/sqlalchemy/sqlalchemy/blob/8503dc2e948908199cd8ba4e6b1d1ddcf92f4020/examples/association/basic_association.py)

▼ Code

```
from datetime import datetime

from sqlalchemy import and_
from sqlalchemy import Column
from sqlalchemy import create_engine
from sqlalchemy import DateTime
from sqlalchemy import Float
from sqlalchemy import ForeignKey
from sqlalchemy import Integer
from sqlalchemy import String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy.orm import Session

Base = declarative_base()

class Order(Base):
    __tablename__ = "order"

    order_id = Column(Integer, primary_key=True)
```

```

customer_name = Column(String(30), nullable=False)
order_date = Column(DateTime, nullable=False, default=datetime.utcnow())
order_items = relationship(
    "OrderItem", cascade="all, delete-orphan", backref="order"
)

def __init__(self, customer_name):
    self.customer_name = customer_name

class Item(Base):
    __tablename__ = "item"
    item_id = Column(Integer, primary_key=True)
    description = Column(String(30), nullable=False)
    price = Column(Float, nullable=False)

    def __init__(self, description, price):
        self.description = description
        self.price = price

    def __repr__(self):
        return "Item(%r, %r)" % (self.description, self.price)

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")
    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
)

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

▼ Documentation

```
"""Illustrate a many-to-many relationship between an
"Order" and a collection of "Item" objects, associating a price
with each via an association object called "OrderItem"
```

The association object pattern is a form of many-to-many which associates additional data with each association between parent and child objects.

The example illustrates an "order", referencing a collection of "items", with a particular price paid associated with each item.

```
"""
```

Prompt

As a code documentation assistant, you are responsible for documenting at the file/script level. When provided a file level code, your approach involves generating a header comment to be added at the top of the file. This comment should be the documentation for the code and include all relevant information needed to understand or use the script. The code is provided below.

Responses

ChatGPT

```
"""
File: main.py
Author: [Your Name]
Description: This script defines SQLAlchemy models and demonstrates their usage.

Dependencies:
    - Python 3.x
    - SQLAlchemy

Usage:
    - Run this script to create an SQLite database and perform operations.

Classes:
    - Order: Represents an order with customer information, order date, and total amount.
    - Item: Represents an item with a description and price.
    - OrderItem: Represents the association between an order and its items.

Example Usage:
    - Creating items, orders, and order items.
    - Querying orders and items.

Note:
    - Make sure to install the required dependencies before running the script.
"""
```

GPT4

```

"""
Title: SQLAlchemy Ordering System

Description:
This script demonstrates the use of SQLAlchemy, a SQL toolkit and

Author Information: [Author's Name, Contact Information]

Date of Creation: [Date]

Usage Instructions:
1. Ensure SQLAlchemy is installed in your Python environment.
2. Run the script directly to see the demonstration of the order
3. Modify the code as needed to adapt to different database sch

Dependencies:
- SQLAlchemy

Input/Output Description:
- Inputs: Demonstrative data for orders and items are hardcoded
- Outputs: The script prints out the details of the orders and

Error Handling:
- Basic error handling is demonstrated in querying operations.
- Additional error handling can be implemented as per requiremen

License Information: [If applicable]

Additional Notes:
- The script uses an in-memory SQLite database, which means the
- This script is intended for educational and demonstration purp
"""

```

Bard

The code provided defines a set of classes that represent an order.

The Order class represents an order placed by a customer. It has

- * order_id: The unique identifier for the order.
- * customer_name: The name of the customer who placed the order.
- * order_date: The date the order was placed.
- * order_items: A list of OrderItem objects that represent the items ordered.

The Item class represents an item that can be ordered. It has the following attributes:

- * item_id: The unique identifier for the item.
- * description: A description of the item.
- * price: The price of the item.

The OrderItem class represents an item that has been ordered. It has the following attributes:

- * order_id: The identifier of the order the item is associated with.
- * item_id: The identifier of the item that was ordered.
- * price: The price of the item at the time it was ordered.

The code also creates an engine and session objects, which are used to interact with the database.

Here is a summary of the code:

- * The code defines three classes: Order, Item, and OrderItem.
- * The code creates an engine and session objects to interact with the database.
- * The code creates four Item objects, one for each type of item.
- * The code creates an Order object and adds three OrderItem objects to it.
- * The code commits the changes to the database.
- * The code queries the database for all orders placed by the customer.

LLama2

```

# File name: orders.py
# Description: Script that defines classes for managing orders
# Author: Your Name
# Date: Current Date

# Classes:
#   Order
#   Item
#   OrderItem

# Functions:
#   None

# Variables:
#   engine
#   Base
#   Session

# Imports:
#   datetime
#   sqlalchemy
#   sqlalchemy.ext.declarative
#   sqlalchemy.orm

# Notes:
#   - This script creates classes for representing orders and items
#   - It also defines relationships between orders and items, and
#   - The script uses SQLAlchemy to interact with the database.
#   - The main function creates a sample order and adds it to the database

```

StarChat

```

"""

```

```

This Python module contains classes that represent data related

```


Classes:

Order - represents a single sales order

Item - represents a product sold by the company

OrderItem - represents the association between an order and one

Functions:

main - creates sample data using the defined classes and queries:
"""

```
import datetime
```

```
from typing import List
```

```
from sqlalchemy import and_
```

```
from sqlalchemy import Column
```

```
from sqlalchemy import create_engine
```

```
from sqlalchemy import DateTime
```

```
from sqlalchemy import Float
```

```
from sqlalchemy import ForeignKey
```

```
from sqlalchemy import Integer
```

```
from sqlalchemy import String
```

```
from sqlalchemy.ext.declarative import declarative_base
```

```
from sqlalchemy.orm import relationship
```

```
from sqlalchemy.orm import Session
```

```
# Create base class which will serve as parent for other models
```

```
Base = declarative_base()
```

```
class Order(Base):
```

```
    """Represents a single sales order."""
```

```
    __tablename__ = "order"
```

```
    # Define columns for the table
```

```
    order_id = Column(Integer, primary_key=True)
```

```

customer_name = Column(String(30), nullable=False)
order_date = Column(DateTime, nullable=False, default=datetime.now)
order_items = relationship(
    "OrderItem", cascade="all, delete-orphan", backref="order"
)

def __init__(self, customer_name: str):
    self.customer_name = customer_name


class Item(Base):
    """Represents a product sold by the company."""

    __tablename__ = "item"

    # Define columns for the table
    item_id = Column(Integer, primary_key=True)
    description = Column(String(30), nullable=False)
    price = Column(Float, nullable=False)

    def __init__(self, description: str, price: float):
        self.description = description
        self.price = price

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


class OrderItem(Base):
    """Represents the association between an order and one or more items"""

    __tablename__ = "orderitem"

    # Define columns for the join table
    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")

def main():
    engine = create_engine("sqlite:///data.db")
    Base.metadata.create_all(engine)

    session = Session(engine)

    # Create some sample data
    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.extend(
        [OrderItem(mug), OrderItem(crowbar, 10.99), OrderItem(hat, 8.99)]
    )
    session.add(order)
    session.commit()

    # Query the database and print results
    order = session.query(Order).filter_by(customer_name="John Smith").first()
    print(
        [(oi.item.description, oi.price) for oi in order.order_items]
    )

```

```
)

# Print out customers who purchased MySQL Crowbars during the time period
q = session.query(Order).join("order_items", "item")
q = q.filter(and_(Item.description == "MySQL Crowbar", Item.quantity > 0))
print([o.customer_name for o in q])

if __name__ == '__main__':
    main()
```