Table Relationships/Joins in Databases

Abdul Haseeb

BS(AI)-IV

Table Relationships

- Tables can be related with one another via columns that act as a bridge between the tables

- Relationships allow us to avoid duplicate data

Example

- Employee table may be related to location table so that we can know which location an employee works at without the need to copy the same location data in every employee’s record.

Relationships

- Make it easy to change things in one place

- Example:
  - If the location moves to new building, we would be able to update the address once in the location table, and every employee related to that location would show the new address
  - In this way we won’t have to update data in multiple tables

Relationships

- Useful to breakout information from a table we don’t need often.

Relationships

- Related by state name, state column in Census and name column in State_Fact
- We will use this pre-defined relationship

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

State_Fact

name	abbreviation	type
New York	NY	state
Washington DC	DC	capitol
Washington	WA	state

Automatic Joins

- stmnt=select(census.c.pop2008, state_fact.c.abbreviation)
- results=connection.execute(stmnt).fetchall()
- print(results)

- SQL Alchemy automatically adds the right join clause because it is predefined in the database

Join

- Accepts a table and an optional expression which defines how two tables are related.
- The expression is not needed if the relationship is predefined and available via reflection.
- Comes immediately after the select() clause and prior to any where(), order_by() or group_by() caluses.

Example

- Find only those records from the census population 2000 column and sum them, where circuit_court from state fact column was 10

- It means for those states where circuit court was 10, because tables are related via states

Code for Example

- stmnt= select(func.sum(census.c.pop2008))
- stmnt= stmnt.select_from(census.join(state_fact))
- stmnt= stmnt.where(state_fact.c.circuit_court == ‘10’)
- result= connection.execute(stmnt).scalar()
- prin(result)

Joining tables without predefined relationships

- Imagine we want to determine the total population in east 2008 that belongs to East South Central division of the census;

- The population and location live in different table, however this time we don’t have predefined relationship

Defining a join relation using select_from

- stmnt= select(func.sum(census.c.pop2000))
- stmnt= stmnt.select_from(
    census.join(state_fact, census.c.state == state_fact.c.name)

)

stmnt= stmnt.where(state_fact.c.census_division_name == ‘East South Central’

result=connection.execute(stmnt).scalar()

print(result)

Code to Create a Database and Insert a Table

```
from sqlalchemy import inspect,create_engine, MetaData, Table, String, Integer, Boolean,Numeric, Column

# Create engine to create a database
engine = create_engine('sqlite:///mydb.sqlite')

with engine.connect() as connection:
    metadata = MetaData()
    employees=Table('employees', metadata, Column('id',Integer()),
                    Column('name', String(255)),
                    Column('salary', Numeric()),
                    Column('active',Boolean())
                    )
    metadata.create_all(engine)
print(inspect(engine).get_table_names())
```

Code to Insert() Rows to a Database

```
from sqlalchemy import create_engine, MetaData, Table, Column, Integer, String, Numeric, Boolean, insert, select, func

engine = create_engine('sqlite:///mydb.sqlite', echo=False)
metadata = MetaData()

employees = Table('employees', metadata,
    Column('id', Integer, primary_key=True),
    Column('name', String(255)),
    Column('salary', Numeric()),
    Column('active', Boolean())
)

metadata.create_all(engine)
```

Continued...

```
# Use one connection for all operations
with engine.connect() as connection:
    # First insert
    connection.execute(insert(employees), [
        {"id": 1, "name": "Ali", "salary": 5.00, "active": True},
        {"id": 2, "name": "Sara", "salary": 6.50, "active": False}
    ])

    # Second insert
    connection.execute(insert(employees), [
        {"id": 3, "name": "Haseeb", "salary": 4.00, "active": True},
        {"id": 4, "name": "Aisha", "salary": 7.20, "active": True}
    ])

    # Count total rows
    total = connection.execute(select(func.count()).select_from(employees)).scalar()
    print("Total number of rows in 'employees':", total)
```

Updating Data in a Table

- from sqlalchemy import create_engine, MetaData, Table, Column, Integer, String, Boolean, insert, update

- # Step 1: Connect to SQLite database and define metadata
- engine = create_engine('sqlite:///mydb.sqlite')
- metadata = MetaData()

- # Step 2: Define the employee table
- employee = Table(
- 'employee', metadata,
- Column('id', Integer, primary_key=True),
- Column('name', String),
- Column('is_active', Boolean, default=False)
-)

Continued...

- # Step 3: Create the table if it doesn't exist
- metadata.create_all(engine)

- # Step 4: Insert 4 rows
- with engine.connect() as conn:
- conn.execute(insert(employee), [
- {'id': 1, 'name': 'Alice', 'is_active': False},
- {'id': 2, 'name': 'Bob', 'is_active': False},
- {'id': 3, 'name': 'Charlie', 'is_active': False},
- {'id': 4, 'name': 'Diana', 'is_active': False}
-])
- conn.commit()
- print("Inserted 4 employees.")

Continued...

- # Step 5: Update employee with id = 3 to set is_active = True
- with engine.connect() as conn:
- stmt = update(employee).where(employee.c.id == 3).values(is_active=True)
- conn.execute(stmt)
- conn.commit()
- print("Employee with ID 3 set as active.")

Deleting specific rows

```
from sqlalchemy import create_engine, MetaData, Table, delete

# Connect to the database
engine = create_engine('sqlite:///mydb.sqlite')
metadata = MetaData()
metadata.reflect(bind=engine)

# Reference the employee table
employee = metadata.tables['employee']


stmt = delete(employee) .where(employee.c.id==3)

# Execute the delete
with engine.connect() as conn:
    result = conn.execute(stmt)
    conn.commit()
    print(f"Deleted Employees Table")
```

Deleting All rows from a Database

```
from sqlalchemy import create_engine, MetaData, Table, delete

# Connect to the database
engine = create_engine('sqlite:///mydb.sqlite')
metadata = MetaData()
metadata.reflect(bind=engine)

# Reference the employee table
employee = metadata.tables['employee']



stmt = delete(employee)
# Execute the delete
with engine.connect() as conn:
    result = conn.execute(stmt)
    conn.commit()
    print(f"Deleted Employees Table")
```

Drop All Tables

```
from sqlalchemy import create_engine, MetaData, Table, delete,inspect

# Connect to the database
engine = create_engine('sqlite:///mydb.sqlite')
metadata = MetaData()
metadata.reflect(bind=engine)

# Reference the employee table
employee = metadata.tables['employee']


stmt = delete(employee)

# Execute the delete
with engine.connect() as conn:
    print(inspect(engine).get_table_names())
    metadata.drop_all(engine)
    print(inspect(engine).get_table_names())
```