📝 Database Pattern Implementation

🎯 Analisi del Problema

Il problema richiede la creazione di una libreria per interagire con diversi database mantenendo un'interfaccia uniforme. Le caratteristiche chiave sono:

- Selezione dinamica del database
- Interfaccia unificata
- Estensibilità per nuovi database

🎨 Pattern Scelti: Strategy + Factory Method

Ho scelto la combinazione di Strategy e Factory Method perché:

1. Strategy: Permette di definire una famiglia di algoritmi intercambiabili
2. Factory Method: Gestisce la creazione degli oggetti database in modo flessibile

Vantaggi ✅

- Maggiore flessibilità nella selezione del database
- Incapsulamento della logica di creazione
- Facilità di aggiungere nuove strategie
- Separazione tra creazione e utilizzo

Svantaggi ❌

- Maggiore complessità iniziale
- Più classi da gestire
- Overhead di performance minimo

💻 Implementazione

```
from abc import ABC, abstractmethod
from typing import Dict, Any

# Strategy Pattern - Interfaccia strategia
class DatabaseStrategy(ABC):
    @abstractmethod
    def find(self, params: Dict) -> Any:
```

```python
        pass

    @abstractmethod
    def create(self, data: Dict) -> Any:
        pass

# Concrete Strategies
class MongoDBStrategy(DatabaseStrategy):
    def __init__(self, connection_string: str):
        self.connection_string = connection_string
        print(f"Connesso a MongoDB: {connection_string}")

    def find(self, params: Dict) -> Any:
        print(f"MongoDB: Ricerca con parametri {params}")
        return {"result": "MongoDB data"}

    def create(self, data: Dict) -> Any:
        print(f"MongoDB: Creazione record {data}")
        return {"id": "123", "data": data}

class PostgreSQLStrategy(DatabaseStrategy):
    def __init__(self, connection_string: str):
        self.connection_string = connection_string
        print(f"Connesso a PostgreSQL: {connection_string}")

    def find(self, params: Dict) -> Any:
        print(f"PostgreSQL: Ricerca con parametri {params}")
        return {"result": "PostgreSQL data"}

    def create(self, data: Dict) -> Any:
        print(f"PostgreSQL: Creazione record {data}")
        return {"id": "456", "data": data}

# Factory Method Pattern
class DatabaseFactory(ABC):
    @abstractmethod
    def create_database(self, connection_string: str) -> DatabaseStrategy:
        pass

class MongoDBFactory(DatabaseFactory):
    def create_database(self, connection_string: str) -> DatabaseStrategy:
        return MongoDBStrategy(connection_string)

class PostgreSQLFactory(DatabaseFactory):
    def create_database(self, connection_string: str) -> DatabaseStrategy:
        return PostgreSQLStrategy(connection_string)
```

```python
# Context che usa la strategia
class DatabaseContext:
    def __init__(self, strategy: DatabaseStrategy):
        self._strategy = strategy

    def set_strategy(self, strategy: DatabaseStrategy):
        self._strategy = strategy

    def find_data(self, params: Dict) -> Any:
        return self._strategy.find(params)

    def insert_data(self, data: Dict) -> Any:
        return self._strategy.create(data)

# Esempio di utilizzo
def main():
    # Creazione delle factories
    mongo_factory = MongoDBFactory()
    postgres_factory = PostgreSQLFactory()

    # Creazione del context con MongoDB
    mongo_db = mongo_factory.create_database("mongodb://localhost:27017")
    context = DatabaseContext(mongo_db)

    print("\nTest con MongoDB:")
    context.find_data({"user": "mario"})
    context.insert_data({"name": "Mario", "age": 30})

    # Cambio a PostgreSQL
    postgres_db =
postgres_factory.create_database("postgresql://localhost:5432")
    context.set_strategy(postgres_db)

    print("\nTest con PostgreSQL:")
    context.find_data({"user": "luigi"})
    context.insert_data({"name": "Luigi", "age": 25})

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

🎬 Output dell'implementazione

```
Connesso a MongoDB: mongodb://localhost:27017
```

```
Test con MongoDB:
MongoDB: Ricerca con parametri {'user': 'mario'}
MongoDB: Creazione record {'name': 'Mario', 'age': 30}

Connesso a PostgreSQL: postgresql://localhost:5432

Test con PostgreSQL:
PostgreSQL: Ricerca con parametri {'user': 'luigi'}
PostgreSQL: Creazione record {'name': 'Luigi', 'age': 25}
```

🔍 Spiegazione dei Pattern

Strategy Pattern

- Definisce una famiglia di algoritmi (strategie di database)
- Rende gli algoritmi intercambiabili
- Permette la selezione dell'algoritmo a runtime

Factory Method Pattern

- Incapsula la creazione degli oggetti database
- Permette di estendere facilmente con nuovi tipi di database
- Mantiene il codice pulito e organizzato

Il flusso delle operazioni è:

1. La factory crea l'implementazione appropriata del database
2. Il context utilizza la strategia selezionata
3. Le operazioni vengono eseguite attraverso l'interfaccia comune