SQL per le applicazioni

Call Level Interface (CLI)

DBMG

1

Call Level Interface

- Le richieste sono inviate al DBMS per mezzo di funzioni del linguaggio ospite
  - soluzione basata su interfacce predefinite
    - API, Application Programming Interface
  - le istruzioni SQL sono passate come parametri alle funzioni del linguaggio ospite
  - non esiste il concetto di precompilatore
- Il programma ospite contiene direttamente le chiamate alle funzioni messe a disposizione dall'API

DBMG 2

2

Call Level Interface

- Esistono diverse soluzioni di tipo Call Level Interface (CLI)
  - standard SQL/CLI
  - ODBC (Open DataBase Connectivity)
    - Standard ampiamente adottato da diversi produttori
  - JDBC (Java Database Connectivity)
    - soluzione per il mondo Java
  - OLE DB
  - ADO
  - ADO.NET
    - soluzione proprietaria Microsoft di SQL/CLI

DBMG 3

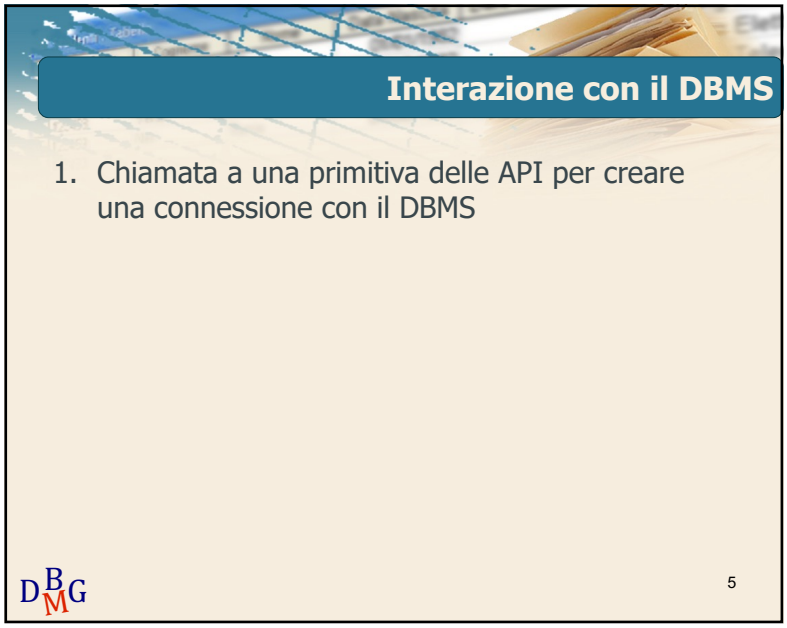
3

Modalità d'uso

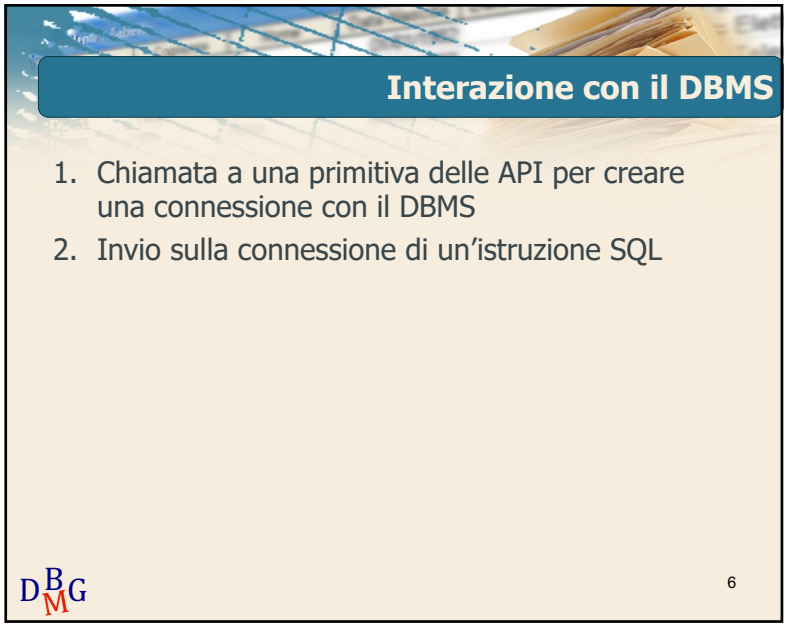
- Indipendentemente dalla soluzione CLI adottata, esiste una strutturazione comune dell'interazione con il DBMS
  - apertura della connessione con il DBMS
  - esecuzione di istruzioni SQL
  - chiusura della connessione

DBMG 4

4


Interazione con il DBMS
1. Chiamata a una primitiva delle API per creare una connessione con il DBMS
DBMG
5

5


Interazione con il DBMS
1. Chiamata a una primitiva delle API per creare una connessione con il DBMS
2. Invio sulla connessione di un'istruzione SQL
DBMG
6

6

Interazione con il DBMS

1. Chiamata a una primitiva delle API per creare una connessione con il DBMS
2. Invio sulla connessione di un'istruzione SQL
3. Ricezione di un risultato in risposta all'istruzione inviata
   - nel caso di SELECT, di un insieme di tuple

DBMG 7

7

Interazione con il DBMS

1. Chiamata a una primitiva delle API per creare una connessione con il DBMS
2. Invio sulla connessione di un'istruzione SQL
3. Ricezione di un risultato in risposta all'istruzione inviata
   - nel caso di SELECT, di un insieme di tuple
4. Elaborazione del risultato ottenuto
   - esistono apposite funzioni per leggere il risultato

DBMG 8

8

Interazione con il DBMS

1. Chiamata a una primitiva delle API per creare una connessione con il DBMS
2. Invio sulla connessione di un'istruzione SQL
3. Ricezione di un risultato in risposta all'istruzione inviata
   - nel caso di SELECT, di un insieme di tuple
4. Elaborazione del risultato ottenuto
   - esistono apposite funzioni per leggere il risultato
5. Chiusura della connessione al termine della sessione di lavoro

DBMG 9

9

Interazione con il DBMS

- ODBC (Open DataBase Connectivity)
  - Metodo di accesso standard verso una base dati
  - Scopo: rendere il protocollo di accesso al database indipendente dal tipo di database utilizzato
  - Python mette a disposizione del programmatore una libreria che consente di accedere via ODBC ad una base dati
- Metodi di accesso mirati ad un DBMS specifico
  - MySQL, Postgres, Microsoft SQL server, …
  - Python mette a disposizione del programmatore librerie specifiche per gran parte dei DBMS

DBMG 10

10

SQL per le applicazioni

Funzioni SQLAlchemy per Flask

DBMG

11

Funzioni SQLAlchemy

- SQLAlchemy è una Libreria Python che permette che consente di interfacciarsi a DB in modo efficiente
  - https://www.sqlalchemy.org/
- Funzionalità supportate
  - Connessione al DB
  - Esecuzione immediata di query SQL
  - Acquisizione e lettura di dati
  - Query multiple e transazioni

DBMG 12

12

Creazione di una connessione

- Chiamata alla funzione create_engine()
  - Starting point delle applicazioni che utilizzano SQLAlchemy, permette di specificare i dettagli della connessione
- Richiede cinque parametri
  - dialect: nome del linguaggio che verrà utilizzato per la connessione
  - username: nome dell'utente nel db
  - password: password dell'utente
  - host: nome della macchina che ospita il DBMS
  - dbname: nome del DB
- Restituisce un identificativo di connessione

```
from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker

dialect = "mysql"
username="root"
password=""
host="127.0.0.1"
dbname = "Opere"
#Connection object creation
engine = create_engine("%s://%s:%s@%s/%s"%(dialect,username,password,host,dbname))
```

DBMG 13

13

Connessione al DB

- Chiamata alla funzione connect()
  - Quando invocata SQLAlchemy effettua la connessione con il DB
  - Utilizza l'identificativo di connessione creato dalla create_engine()
- Restituisce un identificativo di connessione
  - In caso di successo restituisce una connessione attiva
  - In caso di fallimento solleva un'**eccezione**

```
#Establish DB connection
con = engine.connect()
```

DBMG 14

14

Gestione Errori

- Esempio con controllo di eventuali errori di connessione
  - Try: istruzioni da eseguire sempre
  - Except: istruzioni da eseguire in caso di **eccezioni** durante l'esecuzione delle istruzioni all'interno del try
  - SQLAlchemyError: permette di ottenere una stringa con l'errore da visualizzare

```
from sqlalchemy import create_engine
from sqlalchemy.exc import SQLAlchemyError

dialect = "mysql"
username="root"
password=""
host="127.0.0.3"
dbname = "Opere"
#Connection object creation
engine = create_engine("%s://%s:%s@%s/%s"%(dialect,username,password,host,dbname))


try:
    con = engine.connect()

except SQLAlchemyError as e:
  error = str(e.__dict__['orig'])
```

```
(2003, "Can't connect to MySQL server on '127.0.0.3:3306' (60)")
```

```
host="127.0.0.1"
dbname = "Opere2"
```

```
(1049, "Unknown database 'Opere2'")
```

15

15

Chiusura di una connessione

- Deve essere eseguita quando non è più necessario interagire con il DBMS
  - Chiude il collegamento con il DBMS e rilascia le relative risorse
- Chiamata alla funzione close()
  - Utilizza l'identificativo della connessione restituito dalla funzione connect()

```
#Close the DB connection
con.close()
```

16

16

Esecuzione di istruzioni SQL

- Esecuzione immediata dell'istruzione
  - Il server compila ed esegue immediatamente l'istruzione SQL ricevuta

17

Esecuzione immediata

- Chiamata alla funzione execute()
  - Utilizza l'identificativo della connessione restituito dalla funzione connect()
  - Richiede come parametro la query da eseguire, in formato stringa
  - In caso di successo restituisce il risultato della query, in caso di insuccesso solleva un'eccezione
- Esempio:

```
#QUERY SQL
query="SELECT autore.cognome, opera.nome\
       FROM autore, opera\
       WHERE autore.coda = opera.autore"
result = con.execute(query)
```


18

Lettura del risultato SQLAlchamy

- Il risultato della funzione execute() viene memorizzato in una variabile di tipo "cursor"
  - Una variabile speciale, che contiene il risultato dell'interrogazione
  - Per le tabelle è possibile conoscerne l'intestazione utilizzando la funzione keys() sul risultato
- La lettura del risultato avviene riga per riga tramite un cursore

NomeF	NSoci
Andrea	2
Gabriele	2

← Intestazione
← Cursore

DBMG 19

19

Lettura del risultato Jinja2

- Il risultato viene passato a Jinja2 per la visualizzazione con un vettore composto da righe
  - È possibile iterare sulle righe come dei vettori
- Ogni riga è codificata come **una tupla** di valori rappresentanti gli attributi richiesti nella SELECT
  - È possibile leggere le tuple sia come
    - vettore

```
{% for opera in values %}
<tr>
    {% for field in opera %}
    <td >{{ field }}</td>
    {% endfor %}
</tr>
```

    - dizionario

```
{% for opera in values %}
<tr>
    <td >{{ opera["cognome"] }}</t
    <td >{{ opera["nome"] }}</td>
</tr>
{% endfor %}
```

DBMG 20

20

Visualizzazione del risultato

- È possibile passare a jinja2 diversi vettori per indicare l'**intestazione** della tabella ed il **contenuto**

```
try:
    con = engine.connect()

    #QUERY SQL
    query="SELECT autore.cognome, opera.nome\
           FROM autore, opera\
           WHERE autore.coda = opera.autore"
    result = con.execute(query)
    header = result.keys()
    return render_template('opere.html', annoDa=annoDa, annoA=annoA, citta=citta, header=header, values=result)
except SQLAlchemyError as e:

    error = str(e.__dict__['orig'])
    return render_template('errore.html', error_message=error)
```

```
<table>
    <tr>
        {% for field in header %}
        <td >{{ field }}</td>
        {% endfor %}
    </tr>
    {% for opera in values %}
    <tr>
        {% for field in opera %}
        <td >{{ field }}</td>
        {% endfor %}
    </tr>
    {% endfor %}
</table>
```

cognome	nome
Bernini	Apollo e Dafne
Bernini	Baldacchino S.Pietro
Bernini	Fontana dei fiumi
Borromini	S.Ivo la Sapienza

DBMG 21

21

Le transazioni

- Le connessioni avvengono implicitamente in modalità auto-commit
  - Dopo l'esecuzione con successo di ogni istruzione SQL, è eseguito automaticamente commit
- Quando è necessario eseguire commit solo dopo aver eseguito con successo una sequenza di istruzioni SQL
  - Il commit deve essere gestito in modo non automatico
  - Si esegue un solo commit alla fine dell'esecuzione di tutte le istruzioni

DBMG 22

22

Gestione delle transazioni

- Chiamata alla funzione begin()
  - Quando invocata SQLAlchemy inizializza una transazione e disaliblità l'autocommit
  - In caso di successo restituisce una transazione attiva
  - In caso di fallimento solleva un **eccezione**
  - Utilizza l'identificativo della connessione restituito dalla funzione connect()

```
#Initialize a new transaction
trans = con.begin()
```

DBMG 23

23

Gestione delle transazioni

- Se si disabilita l'autocommit le operazioni di commit e rollback devono essere richieste esplicitamente
  - Utilizzano l'identificativo della transazione restituito dalla funzione begin()
- commit ()

```
#Commits the operations
trans.commit()
```

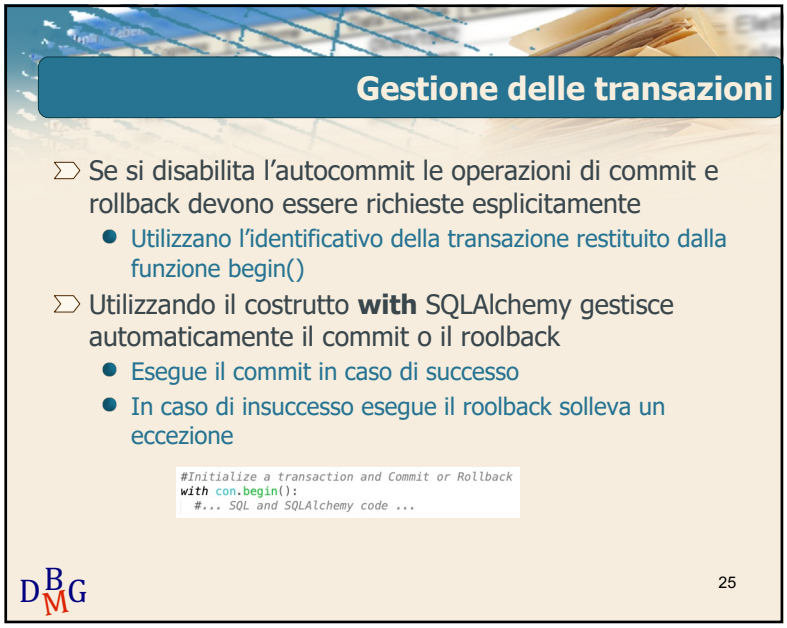
  - Esegue il commit della transazione corrente
  - In caso di insuccesso solleva un eccezione
- rollback ()

```
#Roolback the operations
trans.rollback()
```

  - Esegue il rollback della transazione corrente
  - In caso di insuccesso solleva un eccezione

DBMG 24

24


Gestione delle transazioni
Se si disabilita l'autocommit le operazioni di commit e rollback devono essere richieste esplicitamente
Utilizzano l'identificativo della transazione restituito dalla funzione begin()
Utilizzando il costrutto with SQLAlchemy gestisce automaticamente il commit o il roolback
Esegue il commit in caso di successo
In caso di insuccesso esegue il roolback solleva un eccezione
#Initialize a transaction and Commit or Rollback
with con.begin():
#... SQL and SQLAlchemy code ...
DBMG
25

25