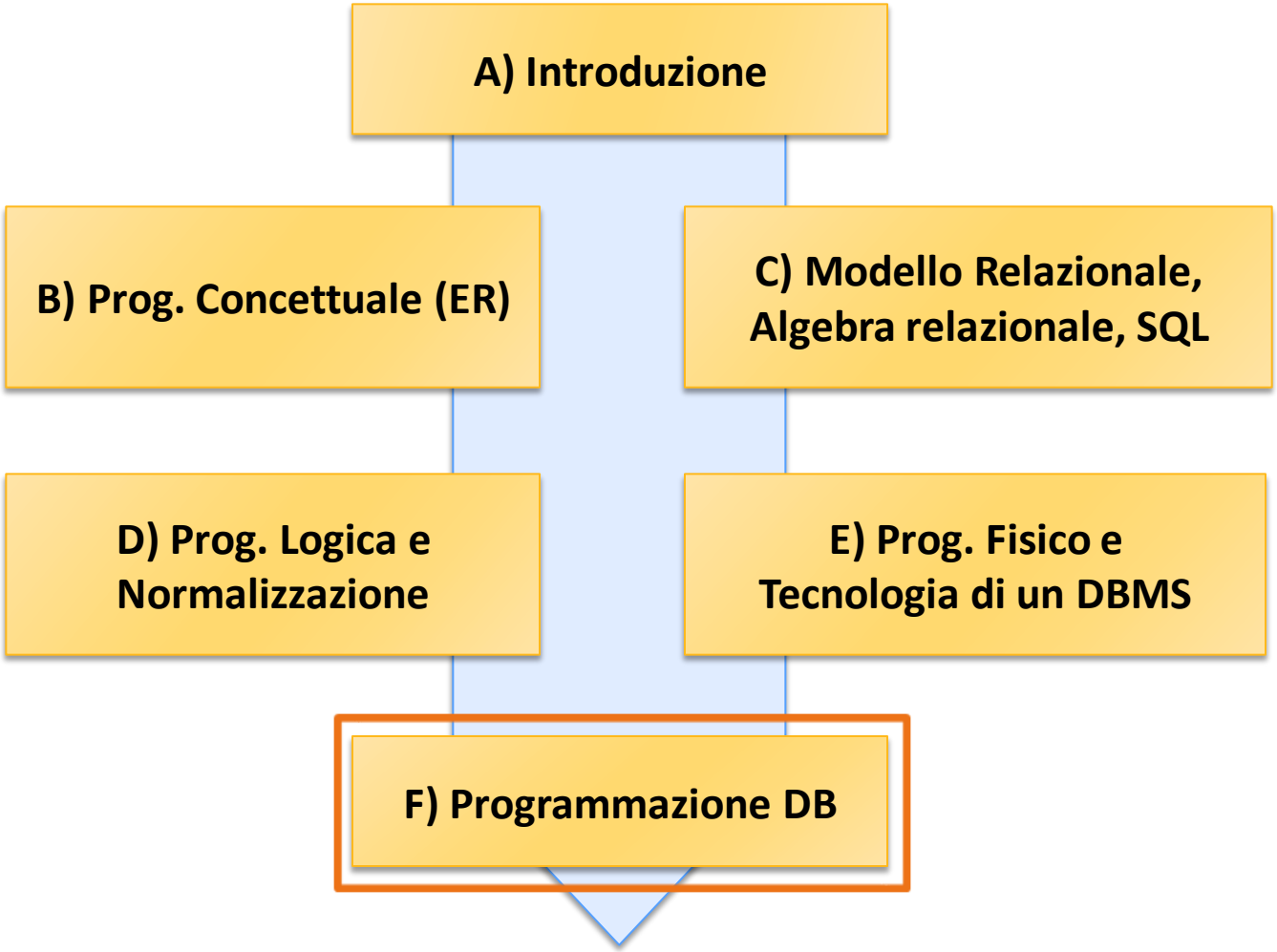
Basi di Dati

Connettività database

Basi di Dati – Dove ci troviamo?



Database e Python

Python supporta l'accesso a tutti i più noti DBMS

E' sufficiente scaricare e installare un modulo adeguato

Noi utilizzeremo MySQL MySQLdb – un modulo DB API 2 compatibile

Python Database API specification

La libreria MySQLdb implementa la standard Python Database API Specification

La API (Application Programming Interface) definisce l'insieme di funzioni a nostra disposizione

Praticamente tutti i moduli database per Python implementano questa API

Quindi, tutti i concetti di base che impareremo sono validi per MySQL ma…

…a meno di qualche piccola modifica sono validi per interfacciarsi a qualunque DBMS

L’importanza di una API

In Python, fino a poco tempo fa, la gestione dei database era un miscuglio di moduli scritti da diverse persone in tutto il mondo

- Mancanza di uniformità tale da poter semplificare la scrittura di codice!

Per semplificare l'accesso ai database relazionali è stata realizzata un' API che è di riferimento per tutti quelli che dovranno scrivere un modulo per l'accesso ad un database relazionale.

Versione corrente: 2.0

L’importanza di una API

L'API mette le basi per le chiamate al modulo dell'accesso al db

Ad esempio: per collegarsi ad Oracle o a MySQL il metodo per farlo si chiama sempre connect()

Questo permette semplicemente di scrivere codice portabile su altre piattaforme database

Connessione al Database

Attraverso il metodo connect(), Python crea un oggetto connessione al db e ne restituisce l'istanza

cursor()

restituisce un cursore dove poter lanciare SELECT, INSERT, UPDATE, stored procedure ecc...

Connessione al Database

commit()

conferma tutte le modifiche apportate al db, disponibile solamente su database di tipi transazionale come ad esempio Oracle e PostgreSQL (MySQL supporta le transazioni solamente con tabelle di tipo InnoDB e non su quelle stabdard MyISAM)

rollback()

revoca tutte le modifiche apportate al db, disponibile solamente su database di tipi transazionale

close()

chiude la connessione con il database

Cursori Database

Per qualsiasi operazione sul DB dobbiamo creare un oggetto Cursor con il metodo cursor()

L'oggetto Cursor creato avrà questi metodi e attributi:

execute(operazione [,parametri])

prepara ed esegue una operazione sul database

callproc(nomeprocedura [,parametri])

chiama una stored procedure (disponibile solamente se il database le supporta)

Cursori Database

Per qualsiasi operazione sul DB dobbiamo creare un oggetto Cursor con il metodo cursor()

L'oggetto Cursor creato avrà questi metodi e attributi:

fetchone()

restituisce il prossimo risultato

fetchmany([size=cursor.arraysize])

restituisce il prossimo insieme di (size) risultati

fetchall()

restituisce tutti i risultati rimanenti

Cursori Database

Per qualsiasi operazione sul DB dobbiamo creare un oggetto Cursor con il metodo cursor()

L'oggetto Cursor creato avrà questi metodi e attributi:

rowcount

attributo che contiene il numero totale di record che ha restituito l'ultima chiamata al metodo execute()

close()

chiude il cursore

Esempio di utilizzo con MySQL

Prima di tutto si importa il modulo:

```
>>>import MySQLdb
```

Poi si crea la connessione, ad esempio:

```
>>>conn = MySQLdb.connect
                    (host="localhost”,
                     user="root”,
                     passwd="”,
                     db="DBprova" )
```

In questo caso: connessione ad un server MySQL che sta girando sulla stessa macchina del nostro script, sulla porta standard 3306, collegamento come utente root senza password

Esempio di utilizzo: SELECT

Creiamo un oggetto di tipo cursore:

```
>>>cursore = conn.cursor()
```

A questo punto possiamo lanciare una query, ad esempio:

```
>>>cursore.execute('SELECT * FROM frutta’)
>>>cursore.fetchall()
>>>((1, 'Mele', 50), (2, 'Pere', 25), (3, 'Banane', 60))
```

Esempio di utilizzo: SELECT

NOTA:

- il risultato è una tupla che contiene tante tuple per ogni record recuperato (nel nostro esempio, tutti i record presenti nella tabella frutta)
- Notate che i dati contenuti nelle tuple sono di tipo diverso: interi, stringhe, ecc.
- Infatti, i tipi di dati SQL vengono, nel limite delle possibilità, tradotti in tipi equivalenti di Python

Esempio di utilizzo: SELECT

E’ anche possibile iterare in un ciclo for i risultati, per elaborarli

Ad esempio, per formattarli in colonne:

```
>>>for record in cursore.fetchall():
...    print '%d\t%s\t%d' % (record[0], record[1], record[2])
...
1    Mele      50
2    Pere      25
3    Banane    60
>>>
```

Esempio di utilizzo: INSERT

Per effettuare una modifica o un inserimento:

```
>>>cursore.execute('INSERT INTO frutta (nome, quantita) VALUES(%s, %s)', ('Pompelmi', 10))
```

Come si vede, si passano due parametri al metodo:

- la stringa SQL di preparazione
- una tupla con i dati da inserire

Questo metodo risulta molto potente per il trattamento veloce dei dati da inserire

Indipendentemente dal tipo di dato, la query SQL completa verrà creata dal metodo execute prima di essere eseguita sul database

Esempio di utilizzo: UPDATE

Allo stesso modo, è possibile effettuare un aggiornamento:

```
>>>cursore.execute('UPDATE frutta SET quantita=%s WHERE nome=\'Pere\'', (5))
```

ATTENZIONE: alla fine delle operazioni, ricordarsi sempre di chiudere gli oggetti relativi (nell'ordine inverso in cui erano stati aperti)! Ad esempio:

```
>>>cursor.close ()
conn.close ()
```