OPEN DATA E DBMS

INTERROGAZIONI
INTEGRAZIONE SQL - PYTHON

SISTEMI INFORMATIVI E DBMS

CORSO DI LAUREA MAGISTRALE IN
MANAGEMENT E MONITORAGGIO DEL TURISMO SOSTENIBILE

PROF. ANDREA PINNA

AA 2019/2020

INTERROGAZIONI

Sono chiamate interrogazioni (query) le istruzioni inviate al DBMS al fine di ottenere una porzione dei dati dell'istanza di un database.

Nel linguaggio SQL la parola da utilizzare per definire una interrogazione è SELECT (selezionare)

INTERROGAZIONI

Per definire l'interrogazione si utilizzano un insieme di parole del linguaggio. Tra queste: FROM. WHERE ...

```
SELECT [cosa mostrare] FROM [da dove] WHERE [condizioni]
```

Un'interrogazione si può leggere come:

"Selezionare [questi attributi] da questa [tabella] quando i dati soddisfano queste [condizioni]"

Il risultato sarà un set (un insieme) di risultati dove ciascun risultato ha per attributi gli attributi selezionati. I risultati comprenderanno i dati delle tabelle scelte che soddisfano le condizioni imposte.

INTERROGAZIONI

Esempio 1 : voglio il nickname di tutti i giocatori con punteggio superiore a 10.

```
SELECT nickname FROM giocatori WHERE punteggio > 10;
```

Con questa interrogazione ottengo un set di dati caratterizzati da un solo attributo (nickname)

Esempio 2 : voglio il numero di ispezioni e il nominativo delle località con codice = 'abc123'

```
SELECT (numeroIspezioni, nominativo) FROM localita WHERE
codice = 'abc123';
```

INTERROGAZIONI

Se si vogliono selezionare tutti gli attributi è possibile utilizzare il carattere * (asterisco) al posto della lista di attributi.

Esempio: restituisci il set con tutti gli attributi della tabella giocatori nella quale il valore di punteggio è maggiore di 10.

```
SELECT * FROM giocatori WHERE punteggio > 10;
```

INTERROGAZIONI

Le clausole del WHERE possono essere combinate.

SQL AND, OR and NOT Operators

```
SELECT * FROM States
WHERE Country='Germany' AND (City='Berlin' OR
City='München');
```

FUNZIONI

Il parametro del SELECT può essere una funzione.

Tra queste vi sono:

- COUNT: restituisce il numero di righe
- SUM: calcola la somma del contenuto di una colonna
- AVG: calcola la media dei valori contenuti in una colonna.

Sintassi comune:

```
SELECT NOME_FUNZIONE(nomeColonna)
FROM nomeTabella
WHERE condizioni;
```

FUNZIONI

ESEMPIO 1: dammi il numero di giocatori che hanno punteggio superiore a 10.

```
SELECT COUNT(codice) FROM giocatori WHERE punteggio > 10;
```

ESEMPIO 2: Voglio sapere la media dei punteggi di tutti i giocatori.

```
SELECT AVG(punteggio) FROM giocatori;
```

(ATTENZIONE: i valori NULL non vengono considerati)

FUNZIONI - TEMPO

La data attuale è contenuta nella variabile globale current_date di tipo DATE.

```
SELECT current_date;  //restituisce la data attuale.
```

Posso estrarre i vari elementi di una data usando ESTRACT

Esempio: voglio l'anno della data attuale.

```
SELECT EXTRACT(year FROM current_date);
```

Posso scegliere year, month e day

Al posto della data attuale può esserci qualunque data valida.

FUNZIONI - TEMPO

SQL include numerose funzioni che permettono di manipolare le date e di formattare i dati temporali.

Ad esempio, la funzione `dayname` prende in ingresso una data e restituisce il nome del giorno.

Esempio 1: Voglio sapere che giorno è oggi:

```
SELECT dayname(current_date);
```

Esempio 2: Voglio sapere che giorno sarà Natale quest'anno:

```
SELECT dayname("2019-12-25");
```

FUNZIONI - TEMPO

Il valore dell'istante temporale attuale è registrato nella variabile current_time di tipo TIME.

```
SELECT (current_time); // vediamo il formato in uscita
'14:04:36'
```

Anche in questo caso posso estrarre i vari elementi della marcatura temporale usando extract. Si possono estrarre hour, minute e second Ad esempio visualizziamo solo i secondi:

```
SELECT extract(second FROM current_time);
```

DISTINCT

La parola chiave DISTINCT serve ad eliminare eventuali elementi doppi da un set di dati.

Esempio 1: Voglio conoscere tutti i nickname dei giocatori senza doppioni

```
SELECT DISTINCT nickname FROM giocatori;
```

Esempio 2: conta quanti nomi dei giocatori diversi ci sono nella tabella.

```
SELECT COUNT(DISTINCT nickname) FROM giocatori;
```

GROUP BY

Con l'istruzione GROUP BY si possono ottenere più set di valori, ed ogni set raggruppa i dati secondo il criterio scelto.

L'argomento di GROUP BY è l'attributo della tabella che si sceglie come parametro del raggruppamento.

Ogni raggruppamento è come un set a sé. Questo significa che eventuali funzioni di conteggio, somma o media verranno fatte su ogni diverso raggruppamento.

GROUP BY

Esempio: voglio conoscere quante volte viene usato ciascun nickname nella tabella giocatori. In questo caso devo raggruppare in base al nickname:

```
SELECT count(nickname), nickname FROM giocatori GROUP BY nickname
```

Il risultato sarà un set che ha per righe il numero di elementi contati dal raggruppamento e il nome del nickname corrispodente.

ORDER BY

Con ORDER BY si può decidere in base a quale attributo ordinare i risultati in uscita dall'interrogazione.

Esempio: mostrare il nickname e il codice dei diversi giocatori in ordine alfabetico rispetto al nikcname.

```
SELECT nickname, codice FROM giocatori ORDER BY nickname;
```

ORDER BY è l'ultima delle istruzioni da specificare.

JOIN

Abbiamo visto che la parola FROM consente di specificare in quale relazione (ovvero la tabella) vi sono i dati che intendo mostrare o elaborare in una query.

In molti casi i dati da elaborare sono contenuti in più tabelle (ad esempio possiamo pensare i casi di entità legate tra loro per mezzo delle associazioni) e questo rende necessaria l'unione dei dati contenuti in esse. L'unione di più tabelle si chiama JOIN.

JOIN

Tra i modi di unire le tabelle si distinguono:

Prodotto Cartesiano (o cross Join)

Join naturale (sul attributo in comune con lo stesso nome)

Join esterno (o outer Join)

JOIN

Per studiare il comportamento dei vari tipi di JOIN usiamo il database di esempio fornito nel materiale didattico.

Aprire o copiare lo script esempio_giocatori.sql su una scheda di Workbench ed eseguire lo script.

Verrà creato un database contenente tre tabelle (giocatori, giochi e partite) ed il loro contenuto.

CROSS JOIN

Il "cross" JOIN tra due tabelle si ottiene facendo corrispondere ad ogni tupla di una tabella ciascuna tupla dell'altra tabella. Se la prima tabella ha m righe e x attributi e la seconda ha n righe e y attributi, la tabella cross join ha m*n righe e x+y attributi.

Si ottiene inserendo due tabelle nel FROM:

```
SELECT [lista_attributi] FROM tabella1, tabella2;
```

Oppure usando la parola JOIN

```
SELECT [lista_attributi] FROM tabella1
JOIN tabella2;
```

CROSS JOIN

Esempio: mostra tutte le combinazioni tra giocatori e partite.

```
SELECT * FROM giocatori,partite;
```

'aac001', 'best', 'noname@hidden.sar', '0', 'aaa001', 'g10001'
'aab001', 'sbam!', 'grace@provider.org', '0', 'aaa001', 'g10001'
'aaa005', 'best', 'best@provider.org', '6', 'aaa001', 'g10001'
'aaa004', 'aaa', 'alpha@game1.abc', '10', 'aaa001', 'g10001'
'aaa003', 'beast', 'beast@greg.com', '11', 'aaa001', 'g10001'
'aaa002', 'slimer', 'ciccio@game1.abc', '0', 'aaa001', 'g10001'
'aaa001', 'pl@yer', 'best@provider.com', '23', 'aaa001', 'g10001'
'aac001', 'best', 'noname@hidden.sar', '0', 'aaa001', 'g10002'
'aab001', 'sbam!', 'grace@provider.org', '0', 'aaa001', 'g10002'
'aaa005', 'best', 'best@provider.org', '6', 'aaa001', 'g10002'
'aaa004', 'aaa', 'alpha@game1.abc', '10', 'aaa001', 'g10002'
'aaa003', 'beast', 'beast@greg.com', '11', 'aaa001', 'g10002'
'aaa002', 'slimer', 'ciccio@game1.abc', '0', 'aaa001', 'g10002'
'aaa001', 'pl@yer', 'best@provider.com', '23', 'aaa001', 'g10002'
'aac001', 'best', 'noname@hidden.sar', '0', 'aaa001', 'g10002'
'aab001', 'sbam!', 'grace@provider.org', '0', 'aaa001', 'g10002'
'aaa005', 'best', 'best@provider.org', '6', 'aaa001', 'g10002'
a', 'alpha@game1.abc', '10', 'aaa001', 'g10002'
ast', 'beast@greg.com', '11', 'aaa001', 'g10002'

NATURAL JOIN

Il natural Join unisce le tabelle in base ad un elemento in comune. Vi sono due modi per ottenerlo.

Il primo è l'utilizzo della parola chiave NATURAL JOIN che prevede l'esistenza di un attributo con lo stesso nome nelle due tabelle.

Sintassi:

```
SELECT [lista_attributi] FROM tabella1 NATURAL
JOIN tabella2;
```

NATURAL JOIN VS INNER JOIN

Lo stesso risultato si può ottenere usando il il WHERE.

```
SELECT [lista_attributi] FROM tabella1, tabella2 WHERE
tabella1.attributo=tabella2.attributo;
```

Questa interrogazione costruisce un cross join e da questo preleva le righe in cui i valori degli attributi corrispondono.

Lo stesso risultato si ottiene con l'INNER JOIN dove però è è possibile specificare su quale attributo eseguire il confronto (non è necessario che abbiano lo stesso nome).

```
SELECT [lista_attributi] FROM tabella1
INNER JOIN tabella2
ON tabella1.attributo=tabella2.attributo;
```

NATURAL JOIN VS INNER JOIN

Esempio: mostrami tutte le coppie nickname e idgioco relative alle partite giocate dai giocatori.

```
SELECT idgioco,nickname FROM giocatori
NATURAL JOIN partite;
```

Oppure:

```
SELECT idgioco,nickname FROM giocatori, partite
WHERE giocatori.codice=partite.codice;
```

Oppure:

```
SELECT idgioco,nickname FROM giocatori INNER JOIN
partite ON giocatori.codice=partite.codice;
```

NATURAL JOIN

```
SELECT * FROM giocatori, partite
WHERE giocatori.codice=partite.codice;
```

'aac001', 'best', 'noname@hidden.sar', '0', 'aaa001', 'g10001'
'aab001', 'sbam!', 'grace@provider.org', '0', 'aaa001', 'g10001'
'aaa005', 'best', 'best@provider.org', '6', 'aaa001', 'g10001'
'aaa004', 'aaa', 'alpha@game1.abc', '10', 'aaa001', 'g10001'
'aaa003', 'beast', 'beast@greg.com', '11', 'aaa001', 'g10001'
'aaa002', 'slimer', 'ciccio@game1.abc', '0', 'aaa001', 'g10001'
'aaa001', 'pl@yer', 'best@provider.com', '23', 'aaa001', 'g10001'
'aac001', 'best', 'noname@hidden.sar', '0', 'aaa001', 'g10002'
'aab001', 'sbam!', 'grace@provider.org', '0', 'aaa001', 'g10002'
'aaa005', 'best', 'best@provider.org', '6', 'aaa001', 'g10002'
'aaa004', 'aaa', 'alpha@game1.abc', '10', 'aaa001', 'g10002'
'aaa003', 'beast', 'beast@greg.com', '11', 'aaa001', 'g10002'
'aaa002', 'slimer', 'ciccio@game1.abc', '0', 'aaa001', 'g10002'
'aaa001', 'pl@yer', 'best@provider.com', '23', 'aaa001', 'g10002'
'aac001', 'best', 'noname@hidden.sar', '0', 'aaa001', 'g10002'
'aab001', 'sbam!', 'grace@provider.org', '0', 'aaa001', 'g10002'
est', 'best@provider.org', '6', 'aaa001', 'g10002'
aa', 'alpha@game1.abc', '10', 'aaa001', 'g10002'
aast', 'beast@greg.com', '11', 'aaa001', 'g10002'

OUTER JOIN

Esegue un Join "esterno" vincolato sulle corrispondenze tra due tabelle di certo attributo. A differenza del natural Join permette di ottenere anche le tuple in cui non si trova una corrispondenza. Si distingue: LEFT, RIGHT e FULL che specificano quale delle tabelle possono avere valori nulli.

Sintassi:

```
SELECT [lista_attributi] FROM tabella1 LEFT OUTER JOIN
tabella2 ON tabella1.attributo1=tabella2.attributo2
```

OUTER JOIN

Esempio: mostrami le coppie codice giocatore e codice gioco delle partite fatte da tutti i giocatori. Mostra anche i giocatori che non hanno mai giocato.

```
SELECT giocatori.codice, idgioco FROM giocatori
LEFT OUTER JOIN partite ON partite.codice=giocatori.codice;
```

ABBREVIAZIONI

Per comodità nel FROM si possono rinominare le tabelle.

Sintassi:

```
SELECT [abbrevviazione].attributo …
FROM  [nomeTabella] [abbrevviazione] …
```

ESEMPIO:

```
SELECT G.codice, P.idgioco FROM giocatori G, partite P
WHERE G.codice=P.codice
```

QUERY ANNIDATE

Il seti di dati ottenuto con il select può essere intepretato come una tabella ed usato nel WHERE.

Come primo caso, vogliamo mostrare gli attributi di una tabella se un certo attributo è contenuto nel risultato di un'altra interrogazione. Uso la parola chiave IN.

```
SELECT [lista_attributi] FROM tabella1
WHERE attributo IN ( SELECT attributo
                     FROM tabella2
                     WHERE Condizioni_attributo )
```

Con la stessa sintassi, al posto di IN posso usare anche altri confronti di tipo booleano.

QUERY ANNIDATE

Come secondo caso vogliamo mostrare tuple solo se esistono o (non esistono) elementi in una interrogazione che coinvolge l'attributo o gli attributi che ci interessano.

```
SELECT [lista_attributi] FROM tabella1
WHERE EXISTS ( SELECT attributo
                      FROM tabella2
                      WHERE Condizioni_attributo )
```

QUERY ANNIDATE

Esempio: mostra nickname e email dei giocatori che non hanno mai giocato partite.

```
SELECT nickname, email  FROM giocatori G
WHERE NOT EXISTS  (SELECT *
                   FROM partite P
                   WHERE  G.codice = P.codice)
```

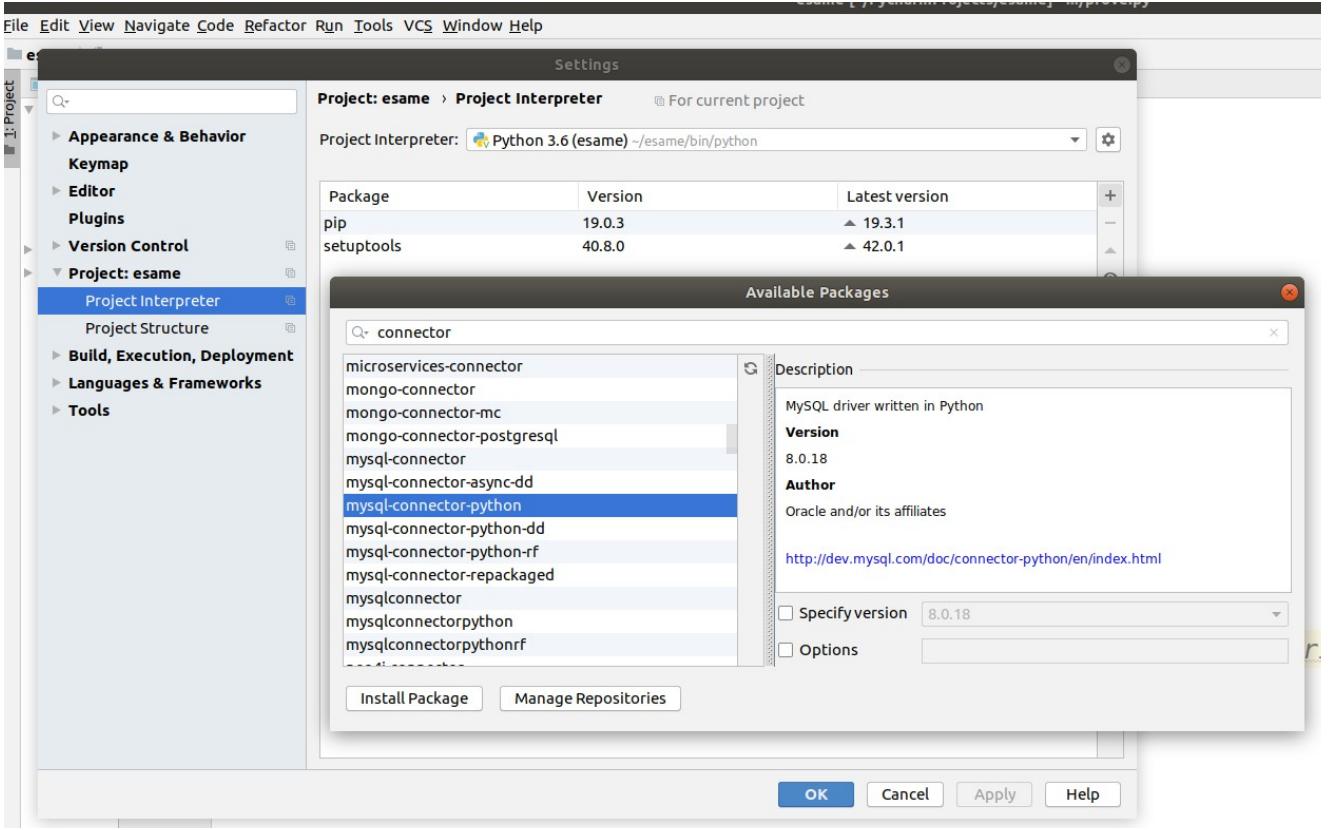
PYTHON MYSQL CONNECTOR

MySQL fornisce una libreria per rendere possibile la comunicazione di un programma Python con un server MySQL in esecuzione nella nostra macchina o in remoto.

La libreria si chiama mysql-connector-python ed è installabile da Pycharm.

PYTHON MYSQL CONNECTOR

Da: File – Settings – Project – Project Interpreter – tastino "+"



CREARE UN NUOVO UTENTE

Se necessario possiamo creare un nuovo utente MYSQL scrivendo uno script SQL su workbench con queste due istruzioni:

```
CREATE USER 'nuovoNomeUtente'@'localhost'
IDENTIFIED BY 'nuovaPassword';
GRANT ALL PRIVILEGES ON *.* TO
'nuovoNomeUtente'@'localhost' WITH GRANT OPTION;
```

Scegliere un nome e una password.

CONNESSIONE PYTHON-MYSQL

Ogni DBMS implementa in modo diverso il sistema di gestione della base di dati. Questo significa che due DBMS relazionali, nonostante entrambi supportino SQL, potrebbero non essere compatibili uno con l'altro. Le istruzioni mandate ad un DBMS (ad esempio MySQL) potrebbero non funzionare su un altro DBMS (PostgreSQL, Oracle, Microsoft ecc.). Questo è evidente quando si vogliono incorporare istruzioni al DBMS all'interno di un linguaggio di programmazione: ogni DBMS può fornire librerie specifiche per specifici linguaggi.

CONNESSIONE PYTHON-MYSQL

In questo corso stiamo usando il DBMS MySQL ed il linguaggio Python e per consentire la comunicazione tra loro è necessario importare una specifica libreria (o modulo) all'interno del programma.

Dopo aver installato l'apposito package possiamo scrivere:

```
import mysql.connector
```

Ora possiamo usare le funzioni della libreria mysql.connector.

CONNECT

La funzione che permette di creare una connessione tra Python e MySQL è connect e fa parte della libreria mysql.connector.

Sintassi:

```
[nome_connettore] = mysql.connector.connect([lista_parametri])
```

Il risultato è una variabile di tipo "connection". In particolare, il tipo del risultato di questa funzione è:

```
<class 'mysql.connector.connection_cext.CMySQLConnection'>
```

CONNECT: ESEMPIO

Creiamo un programma che apre una connessione con il database "gioco" installato sulla nostra macchina.

```
import mysql.connector
connessione = mysql.connector.connect(user='andrea',
                                      password='mysql',database='gioco')
```

CONNECT: PARAMETRI

La funzione connect ha diversi parametri che vanno specificati quando la si chiama. Tra questi ci sono:

```
user : nome utente del database (string)
passwd : la password dell'utente  (string)
database : nome del database da aprire(string)
```

Per accedere ad un database remoto si specificano anche:

```
host : ip del server (default: localhost)
port : porta di comunicazione (default: 3306)
```

CONNECT: METODI

Il connettore (ovvero il risultato della funzione connect) è un oggetto con diversi metodi.

close: Il metodo `close()` permette di **chiudere** la connessione e va chiamato al termine delle operazioni sul database (come con i file).

cursosr: il metodo `cursor()` **restituisce** un "cursore" con il quale è possibile inviare le interrogazioni al database e di ricevere i risultati. È possibile chiudere il cursore chiamando il suo metodo close.

CONNECT: METODI

commit: Il metodo `commit()` permette di **regisrtare** le modifiche sul database.

Se il programma invia istruzioni di modifica al database è necessario renderle effettive con il metodo commit eseguito dopo la richiesta di modifiche.

Esempio:

```
connessione.commit()
```

CONNECT: ESEMPIO CURSORE

Esempio: apriamo il database "gioco" e creiamo il cursore "miocursore". Mostriamo a video il tipo di dato del cursore. Infine chiudiamo il cursore e la connessione.

```
import mysql.connector
connessione = mysql.connector.connect(user='andrea',
                      password='mysql',database='gioco')
miocursore = connessione.cursor()
print(type(miocursore))
miocursore.close()
connessione.close()
```

CURSORE

Il cursore è l'oggetto che ci permette di manipolare il database. Tra i suoi metodi vediamo:

`execute`: prende in ingresso una stringa contenente un'istruzione SQL la invia al DBMS.

`fetchone`: preleva la prima risposta del DBMS e la **restituisce** sotto forma di tupla (o di singolo elemento).

`fetchmany e fetchall`: prelevano le risposte del DBMS e le **restituiscono** sotto forma di lista di tuple (o di singoli elementi).

`close`: chiude il cursore.

CURSORE: EXECUTE

Il metodo execute del cursore invia una istruzione SQL al DBMS

```
miocursore.execute([istruzione SQL])
```

Esempi:

```
miocursore.execute("USE gioco")
miocursore.execute("SELECT current_time")
```

Non restituisce nulla. L'eventuale risultato resta nel DBMS in attesa di essere letto.

CURSORE: FETCHONE

I metodi della famiglia "fetch" consentono di leggere i set di dati risultanti dopo l'utilizzo di execute.

Il metodo `fetchone` non ha argomenti e restituisce il risultato dell'interrogazione quando contiene una sola tupla (o un singolo valore). Se il set risultante contiene più tuple il metodo da errore.

```
[nome_variabile] = [nomecursore].fetchone()
```

Esempio:

```
risultato = miocursore.fetchone()
```

CURSORE: FETCHMANY

Il metodo `fetchmany` ha come argomento il numero massimo di tuple che ci aspettiamo come risposta dal DBMS. Se il set risultante contiene più tuple di quelle previste il metodo da errore. Restituisce una lista di tuple anche se l'iterrogazione restituisce un singolo elemento.

```
[nome_variabile] = [nomecursore].fetchmany(minRighe)
```

Esempio:

```
risultati = miocursore.fetchmany(10) #se i risultati
sono più di 10 da errore.
```

CURSORE: FETCHALL

Il metodo `fetchall` non ha argomenti. Preleva tutti i risultati della query e li restituisce come una lista di tuple.

```
[nome_variabile] = [nomecursore].fetchall()
```

Esempio:

```
risultati = miocursore.fetchall()
```

ESEMPIO D'UTILIZZO

Scrivere un programma che interroga il database gioco per conoscere tutte le righe della tabella giocatori.

```
import mysql.connector
connessione =
mysql.connector.connect(user='andrea',password='mysql',database='g
ioco')
miocursore = connessione.cursor()

miocursore.execute("select * from giocatori")
listaGiocatori = miocursore.fetchall()
print(listaGiocatori)

miocursore.close()
connessione.close()
```

FORMATTAZIONE STRINGHE

Abbiamo visto che le istruzioni SQL sono inviate come stringhe. Per creare interrogazioni con dati generati dinamicamente (ad esempio forniti dall'utente del programma) dobbiamo essere in grado di creare dinamicamente le stringhe di istruzioni SQL.

Sappiamo che le stringhe si possono comporre utilizzando la concatenazione. Esiste un altra modalità che prevede l'uso del metodo format.

FORMATTAZIONE STRINGHE

Il metodo format delle stringhe sostituisce ogni occorrenza dei caratteri “{}” (aperta e chiusa parentesi graffa) con uno degli argomenti passati al metodo. in base all’ordine.

Sintassi:

```
[stringa_con_{}].format(valore1,valore2,... )
```

FORMATTAZIONE STRINGHE

Esempio:

```
conteggio = 10
miaStringa="il {} vale: {}."
modifica=miaStringa.format("dato",conteggio)
print(modifica)
```

La stringa "modifica" è ottenuta sostituendo nell'ordine le coppie {} con i due argomenti del metodo format.

ESERCIZIO 1

Scrivere un programma chiede all'utente di specificare quale attributo si vuole visualizzare.

Il programma crea una istruzione SQL per ottenere quel singolo attributo della tabella giocatori. Il programma recupera il risultato e lo stampa a video.

Gestire l'errore che si ha quando l'interrogazione non va a buon fine (ad esempio se l'attributo non esiste) stampando un messaggio di errore.

Infine stampare a video la stringa "Operazione conclusa".

ESERCIZIO 2

Scrivere un programma che permette all'utente di inserire una nuova riga nella tabella "giochi". L'utente deve specificare l'id del gioco ed il nome del gioco.

Il programma prende quei dati e crea un istruzione SQL da inviare al DBMS (usare il metodo commit del connector per registrare le modifiche nel databse).

Gestire l'errore per evitare che il programma si blocchi, mostrando a video un messaggio all'utente.

Il programma infine mostra a video tutto il contenuto della tabella "giochi".

ESERCIZIO 3

Scrivere un programma che scrive sul file “tabella.txt” tutto il contenuto della tabella giocatori formattato nel modo seguente:

L’utente [codice] si chiama [nickname]. Ha [punti] punti e la sua e-mail è [email].