Linguaggio SQL: costrutti avanzati

SQL per le applicazioni

SQL per le applicazioni

- Introduzione
- Concetto di cursore
- Aggiornabilità
- SQL statico e dinamico
- Embedded SQL
- Call Level Interface (CLI)
- Stored Procedure
- Confronto tra le alternative

SQL per le applicazioni

Introduzione

Prelievo mediante bancomat

- Operazioni svolte
  - verificare la validità del bancomat e del codice PIN
  - selezionare l'operazione di prelievo
  - specificare l'importo richiesto
  - verificare la disponibilità
  - memorizzare il movimento
  - aggiornare il saldo
  - erogare la somma richiesta

Prelievo mediante bancomat

- Per svolgere molte delle operazioni indicate è necessario accedere alla base di dati
  - esecuzione di istruzioni SQL
- Le operazioni devono essere svolte nell'ordine corretto

Prelievo presso uno sportello bancario

- Operazioni svolte
  - verificare l'identità dell'utente
  - comunicare l'intenzione di effettuare un prelievo
  - verificare la disponibilità
  - memorizzare il movimento
  - aggiornare il saldo
  - erogare la somma richiesta

Prelievo presso uno sportello bancario

- Per svolgere molte delle operazioni indicate è necessario accedere alla base di dati
  - esecuzione di istruzioni SQL
- Le operazioni devono essere svolte nell'ordine corretto

Esempio: operazioni bancarie

- Le operazioni bancarie richiedono di accedere alla base di dati e di modificarne il contenuto
  - esecuzione di istruzioni SQL
    - i clienti e il personale della banca non eseguono direttamente le istruzioni SQL
  - un'applicazione nasconde l'esecuzione delle istruzioni SQL
- La corretta gestione delle operazioni bancarie richiede di eseguire una sequenza precisa di passi
  - un'applicazione permette di specificare l'ordine corretto di esecuzione delle operazioni

Applicazioni e SQL

- Per risolvere problemi reali non è quasi mai sufficiente eseguire singole istruzioni SQL
- Servono applicazioni per
  - gestire la logica applicativa
    - flusso di operazioni da eseguire
  - acquisire e gestire i dati forniti in ingresso
    - scelte dell'utente, parametri
  - restituire i risultati all'utente in formati diversi
    - rappresentazione non relazionale dei dati
      - documento XML
    - visualizzazione complessa delle informazioni
      - grafici, report

Integrazione tra SQL e applicazioni

- Le applicazioni sono scritte in linguaggi di programmazione tradizionali di alto livello
  - C, C++, Java, C#, ...
  - il linguaggio è denominato *linguaggio ospite*
- Le istruzioni SQL sono usate nelle applicazioni per accedere alla base di dati
  - interrogazioni
  - aggiornamenti

Integrazione tra SQL e applicazioni

- È necessario integrare il linguaggio SQL e i linguaggi di programmazione
  - SQL
    - linguaggio dichiarativo
  - linguaggi di programmazione
    - tipicamente procedurali

Conflitto di impedenza

- Conflitto di impedenza
  - le interrogazioni SQL operano su una o più tabelle e producono come risultato una tabella
    - approccio set oriented
  - i linguaggi di programmazione accedono alle righe di una tabella leggendole *una a una*
    - approccio tuple oriented
- Soluzioni possibili per risolvere il conflitto
  - uso di cursori
  - uso di linguaggi che dispongono in modo naturale di strutture di tipo "insieme di righe"

SQL e linguaggi di programmazione

- Tecniche principali di integrazione
  - Embedded SQL
  - Call Level Interface (CLI)
    - SQL/CLI, ODBC, JDBC, OLE DB, ADO.NET, ..
  - Stored procedure
- Classificabili in
  - client side
    - embedded SQL, call level interface
  - server side
    - stored procedure

Approccio client side

- L'applicazione
  - è esterna al DBMS
  - contiene tutta la logica applicativa
  - richiede al DBMS di eseguire istruzioni SQL e di restituirne il risultato
  - elabora i dati restituiti

Approccio server side

- L'applicazione (o una parte di essa)
  - si trova nel DBMS
  - tutta o parte della logica applicativa si sposta nel DBMS

Approccio client side vs server side

- Approccio client side
  - maggiore indipendenza dal DBMS utilizzato
  - minore efficienza
- Approccio server side
  - dipendente dal DBMS utilizzato
  - maggiore efficienza

SQL per le applicazioni

Concetto di cursore

Conflitto di impedenza

- Principale problema di integrazione tra SQL e linguaggi di programmazione
  - le interrogazioni SQL operano su una o più tabelle e producono come risultato una tabella
    - approccio set oriented
  - i linguaggi di programmazione accedono alle righe di una tabella leggendole *una a una*
    - approccio tuple oriented

Cursori

- Se un'istruzione SQL restituisce una sola riga
  - è sufficiente specificare in quali variabili del linguaggio ospite memorizzare il risultato dell'istruzione
- Se un'istruzione SQL restituisce una tabella (insieme di tuple)
  - è necessario un metodo per leggere (e passare al programma) una tupla alla volta dal risultato dell'interrogazione
    - uso di un *cursore*

SQL per le applicazioni

SQL statico e dinamico

SQL statico

- Le istruzioni SQL da eseguire sono note durante la scrittura dell'applicazione
  - è nota la definizione di ogni istruzione SQL
  - le istruzioni possono contenere variabili
    - il valore delle variabili è noto solo durante l'esecuzione dell'istruzione SQL

SQL statico

- La definizione delle istruzioni SQL avviene durante la scrittura dell'applicazione
  - semplifica la scrittura dell'applicazione
    - è nota a priori la struttura di interrogazioni e risultati
  - rende possibile l'ottimizzazione a priori delle istruzioni SQL
    - durante la fase di compilazione dell'applicazione, l'ottimizzatore del DBMS
      - compila l'istruzione SQL
      - crea il piano di esecuzione
    - queste operazioni non sono più necessarie durante l'esecuzione dell'applicazione
      - esecuzione più efficiente

SQL dinamico

- Le istruzioni SQL da eseguire *non* sono note durante la scrittura dell'applicazione
  - le istruzioni SQL sono definite dinamicamente dall'applicazione in fase di esecuzione
    - dipendono dal flusso applicativo eseguito
  - le istruzioni SQL possono essere fornite in ingresso dall'utente

SQL dinamico

- La definizione a tempo di esecuzione delle istruzioni SQL
  - permette di definire applicazioni più complesse
    - offre una maggiore flessibilità
  - rende più difficile la scrittura delle applicazioni
    - durante la scrittura non è noto il formato del risultato dell'interrogazione
  - rende l'esecuzione meno efficiente
    - durante ogni esecuzione dell'applicazione, è necessario compilare e ottimizzare ogni istruzione SQL

SQL dinamico

- Se la stessa interrogazione dinamica deve essere eseguita più volte nella *stessa sessione* di lavoro
  - è possibile ridurre i tempi di esecuzione
    - si effettua una sola volta la compilazione e la scelta del piano di esecuzione
    - si esegue l'interrogazione più volte (con valori diversi delle variabili)

SQL per le applicazioni

Embedded SQL

Embedded SQL

- Le istruzioni SQL sono "incorporate" nell'applicazione scritta in un linguaggio di programmazione tradizionale (C, C++, Java, ..)
  - la sintassi SQL è diversa da quella del linguaggio ospite
- Le istruzioni SQL non sono direttamente compilabili da un compilatore tradizionale
  - devono essere riconosciute
    - sono preceduti dalla parola chiave EXEC SQL
  - devono essere sostituite da istruzioni nel linguaggio di programmazione ospite

Precompilazione

- Il precompilatore
  - identifica le istruzioni SQL incorporate nel codice
    - parti precedute da EXEC SQL
  - sostituisce le istruzioni SQL con chiamate a funzioni di una API specifica del DBMS prescelto
    - funzioni scritte nel linguaggio di programmazione ospite
  - (opzionale) invia le istruzioni SQL statiche al DBMS che le compila e le ottimizza
- Il precompilatore è legato al DBMS prescelto

SQL per le applicazioni

Call Level Interface (CLI)

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

Call Level Interface

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

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

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 primitive per leggere il risultato
5. Chiusura della connessione al termine della sessione di lavoro

MySQL Connector/Python

- MySQL Connector/Python consente ai programmi Python di accedere ad una base dati MySQL, attraverso una API

- Linguaggio SQL standard

- Tutorial
  - https://www.w3schools.com/python/python_mysql_getstarted.asp
  - https://dev.mysql.com/doc/connector-python/en/

Connessione al server MySQL

- Il costruttore connect() crea una connessione con il server MySQL e restituisce un oggetto di tipo MySQLConnection
- close() chiude la connessione

```
import mysql.connector
mydb = mysql.connector.connect(
          host="localhost"
          user="yourusername",
          password="yourpassword")

……..

mydb.close()
```

Invio delle istruzioni SQL

- Esempio: creare un DB
  - Il cursore è una struttura che permette di scorrere i record restituiti da una query

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE mydatabase")
```

Invio delle istruzioni SQL

- Esempio: connettersi ad un DB esistente

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
```

Invio delle istruzioni SQL

- Esempio: create una tabella

```
mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))")
```

- Esempio: inserire una tupla
  - È necessario il commit

```
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)

mydb.commit()
```

Invio delle istruzioni SQL

- Esempio: inserire più tuple

```
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = [
  ('Peter', 'Lowstreet 4'),
  ('Amy', 'Apple st 652'),
  ('Hannah', 'Mountain 21'),
  ('Michael', 'Valley 345'),
  ('Sandy', 'Ocean blvd 2'),
  ('Betty', 'Green Grass 1'),
  ('Richard', 'Sky st 331'),
  ('Susan', 'One way 98'),
  ('Vicky', 'Yellow Garden 2'),
  ('Ben', 'Park Lane 38'),
  ('William', 'Central st 954'),
  ('Chuck', 'Main Road 989'),
  ('Viola', 'Sideway 1633')
]

mycursor.executemany(sql, val)
```

Invio delle istruzioni SQL

- Esempio: interrogazione e visualizzazione del risultato
  - Il medoto fetchall() prende tutte le righe dell'ultima istruzione eseguita

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Invio delle istruzioni SQL

- Esempio: interrogazione e visualizzazione del risultato

```
mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address LIKE '%way%'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Invio delle istruzioni SQL

- Esempio: interrogazione e visualizzazione del risultato

```
mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address = %s"
adr = ("Yellow Garden 2", )

mycursor.execute(sql, adr)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

SQL per le applicazioni

Stored Procedure

Stored procedure

- La stored procedure è una funzione o una procedura definita all'interno del DBMS
  - è memorizzata nel dizionario dati
    - fa parte dello schema della base di dati
- È utilizzabile come se fosse un'istruzione SQL predefinita
  - può avere parametri di esecuzione
- Contiene codice applicativo e istruzioni SQL
  - il codice applicativo e le istruzioni SQL sono fortemente integrati tra loro

Stored procedure: linguaggio

- Il linguaggio utilizzato per definire una stored procedure
  - è un'estensione procedurale del linguaggio SQL
  - è dipendente dal DBMS
    - prodotti diversi offrono linguaggi diversi
    - l'espressività del linguaggio dipende dal prodotto prescelto

Stored procedure: esecuzione

- Le stored procedure sono integrate nel DBMS
  - approccio server side
- Le prestazioni sono migliori rispetto a embedded SQL e CLI
  - ogni stored procedure è compilata e ottimizzata *una sola volta*
    - subito dopo la definizione
    - oppure la prima volta che è invocata

Linguaggi per le stored procedure

- Esistono diversi linguaggi per definire stored procedure
  - PL/SQL
    - Oracle
  - SQL/PL
    - DB2
  - Transact-SQL
    - Microsoft SQL Server
  - PL/pgSQL
    - PostgreSQL

Connessione al DBMS

- Non occorre effettuare la connessione al DBMS all'interno di una stored procedure
  - il DBMS che esegue le istruzioni SQL è lo stesso in cui è memorizzata la stored procedure

SQL per le applicazioni

Confronto tra le alternative

Embedded SQL, CLI e Stored procedure

- Le tecniche proposte per l'integrazione del linguaggio SQL nelle applicazioni hanno caratteristiche diverse
- Non esiste un approccio sempre migliore degli altri
  - dipende dal tipo di applicazione da realizzare
  - dipende dalle caratteristiche delle basi di dati
    - distribuite, eterogenee
- È possibile utilizzare soluzioni miste
  - invocazione di stored procedure tramite CLI o embedded SQL

Embedded SQL vs Call Level Interface

- Embedded SQL
  - (+) precompila le interrogazioni SQL statiche
    - più efficiente
  - (-) dipendente dal DBMS e dal sistema operativo usato
    - a causa della presenza del precompilatore
  - (-) generalmente non permette di accedere contemporaneamente a più basi di dati diverse
    - in ogni caso, è un'operazione complessa

Embedded SQL vs Call Level Interface

- Call Level Interface
  - (+) indipendente dal DBMS utilizzato
    - solo in fase di compilazione
      - la libreria di comunicazione (driver) implementa un'interfaccia standard
      - il funzionamento interno dipende dal DBMS
    - il driver è caricato e invocato dinamicamente a runtime
  - (+) non necessita di un precompilatore

Embedded SQL vs Call Level Interface

- Call Level Interface
  - (+) permette di accedere dalla stessa applicazione a più basi di dati
    - anche eterogenee
  - (-) usa SQL dinamico
    - minore efficienza
  - (-) solitamente supporta un sottoinsieme di SQL

Stored procedure vs approcci client side

- Stored procedure
  - (+) maggiore efficienza
    - sfrutta la forte integrazione con il DBMS
    - riduce la quantità di dati inviati in rete
    - le procedure sono precompilate

Stored procedure vs approcci client side

- Stored procedure
  - (-) dipendente dal DBMS utilizzato
    - usa un linguaggio ad hoc del DBMS
    - solitamente non portabile da un DBMS a un altro
  - (-) i linguaggio utilizzati offrono meno funzionalità dei linguaggi tradizionali
    - assenza di funzioni per la visualizzazione complessa dei risultati
      - grafici e report
    - meno funzionalità per la gestione dell'input

Stored procedure vs approcci client side

- Approcci client side
  - (+) basati su linguaggi di programmazione tradizionali
    - più noti ai programmatori
    - compilatori più efficienti
    - maggiori funzionalità per la gestione di input e output
  - (+) in fase di scrittura del codice, maggiore indipendenza dal DBMS utilizzato
    - solo per gli approcci basati su CLI
  - (+) possibilità di accedere a basi di dati eterogenee

Stored procedure vs approcci client side

- Approcci client side
  - (-) minore efficienza
    - minore integrazione con il DBMS
    - compilazione delle istruzioni SQL a tempo di esecuzione
      - soprattutto per approcci basati su CLI