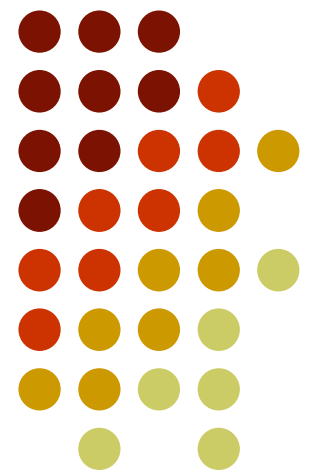


Sviluppo di applicazioni per basi di dati

Elena Ferrari
Basi di Dati
A.A. 2018/2019





Un esempio applicativo

Operazioni bancarie:

- Operazione di prelievo dal proprio conto corrente mediante bancomat





Un esempio applicativo

Operazioni bancarie:

- Operazioni di prelievo dal proprio conto corrente mediante bancomat
- Operazione di prelievo dal proprio conto corrente presso uno sportello bancario



Prelievo bancomat



Operazioni svolte:

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





Prelievo bancomat

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





Operazioni bancarie

- Le operazioni bancarie richiedono di accedere alla base di dati e di modificarne il contenuto tramite SQL
 - i clienti e il personale della banca non eseguono direttamente le istruzioni SQL
 - un'applicazione nasconde l'esecuzione delle istruzioni SQL
- Per risolvere problemi reali non è quasi mai sufficiente eseguire singole istruzioni SQL



Applicazioni ed SQL

Servono applicazioni per:

- acquisire e gestire i dati forniti in ingresso
 - scelte dell'utente, parametri
- gestire la logica applicativa
 - flusso di operazioni da eseguire
- restituire i risultati all'utente in formati diversi
 - rappresentazione non relazionale dei dati (documento XML)
- visualizzare in maniera anche complessa delle informazioni
 - grafici, report



Il problema

- SQL permette di accedere ai dati memorizzati in una base di dati e di aggiornarli
- Per scelte progettuali, il suo potere espressivo è però limitato (rispetto a quello di un generico linguaggio di programmazione)
- Questo significa che non tutte le elaborazioni che possiamo volere applicare ai dati possono essere espresse in SQL
- In particolare, SQL non è:
 - computazionalmente completo, né
 - operazionalmente completo

Completezza computazionale



- E' relativa alla capacità di esprimere nel linguaggio tutte le computazioni teoricamente possibili
- In SQL mancano costrutti tipici dei generici linguaggi di programmazione (imperativi) quali il costrutto di scelta o quello di iterazione



Completezza operativa

E' relativa alla capacità di esprimere nel linguaggio operazioni che richiedono una comunicazione diretta con il sistema operativo e, attraverso di esso, con l'hardware e le periferiche:

- in SQL mancano, ad esempio, costrutti per scrivere in un file, stampare, sviluppare un'interfaccia utente



Soluzione

- E' quindi necessario estendere il potere espressivo di SQL
- Idea: combinare SQL con un linguaggio di programmazione generico:
 - SQL: consente accesso ottimizzato ai dati
 - linguaggio di programmazione: garantisce completezza computazionale ed operativa



Approcci all'integrazione

- Principalmente due:
 - un linguaggio di programmazione esistente (es. C, C++, C#, Java) viene integrato con SQL
 - estensioni procedurali di SQL

Estensione di un linguaggio esistente



- Il codice viene eseguito in un ambiente esterno al DBMS (**client-side**)
- La completezza (computazionale ed operativa) è garantita
- Due approcci principali:
 - **librerie di funzioni**
 - **SQL ospitato – embedded SQL**



Librerie di funzioni

Viene resa disponibile per il linguaggio una libreria di funzioni (API) che definisce l'interfaccia di comunicazione tra il DBMS ed il linguaggio di programmazione (linguaggio ospite):

- funzioni per la connessione alla base di dati e l'esecuzione di comandi SQL
- oltre alle librerie proprietarie dei vari DBMS (CLI), esistono anche librerie standard, che garantiscono interoperabilità (es. ODBC, JDBC) ed SQL/CLI



SQL ospitato

- I comandi SQL vengono utilizzati direttamente all'interno di un linguaggio di programmazione (detto linguaggio ospite)
- Uno speciale pre-compilatore, dato un programma nel linguaggio esteso:
 - produce un programma interamente scritto nel linguaggio ospite
 - i comandi SQL vengono sostituiti con chiamate al DBMS tramite CLI standard o proprietarie

Estensioni procedurali di SQL



- SQL viene esteso con gli usuali costrutti dei linguaggi di programmazione
- Può essere utilizzato all'interno del DBMS, per definire funzioni e procedure richiamabili da qualunque applicazione
- Lo standard SQL propone un'estensione procedurale (SQL/PSM), ancora non del tutto recepito dai DBMS commerciali
- Soluzione server-side

Estensioni procedurali di SQL



- Oracle: PL/SQL
- PostGres SQL: PLpgSQL
- Microsoft SQL Server: T(ransact)-SQL
- DB2: SQL PL (il più vicino a SQL/PSM insieme al linguaggio fornito da MySQL)



Client vs server side

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



Approcci all'integrazione

- L'approccio client side (più diffuso) è scelta ovvia quando applicazioni esistenti devono essere estese per interagire con un DBMS
- L'approccio server side è utilizzato da nuove applicazioni che richiedono una forte interazione con il DBMS
- Approccio più diffuso: combinazione dei due
 - con le estensioni procedurali di SQL, si definiscono procedure e funzioni relative alle operazioni più frequenti sulla base di dati
 - tali procedure e funzioni sono poi richiamate da applicazioni sviluppate con un linguaggio tradizionale



Integrazione

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



Impedence mismatch

- Dovuto alle differenze tra i due linguaggi
- Due differenze principali:
 - differenze nei tipi di dato: necessità di binding tra i tipi di dato del linguaggio e quelli di SQL (es. VARCHAR – String per Java)
 - differenze nella modalità di elaborazione: set-oriented per SQL, tuple-oriented per linguaggi di programmazione



Flusso di esecuzione

Qualunque sia l'approccio scelto, il programma deve eseguire i seguenti passi fondamentali:

1. connessione alla base di dati
2. esecuzione dei comandi SQL
3. chiusura della connessione



Esecuzione di comandi SQL

Per ogni comando SQL, l'interazione tra il linguaggio di programmazione ed il DBMS coinvolge tre passi:

- **preparazione del comando:** generazione delle strutture dati necessarie per la comunicazione con il DBMS ed eventuale compilazione ed ottimizzazione del comando da parte del DBMS
- **esecuzione del comando:** il comando viene eseguito utilizzando le strutture dati e le informazioni generate durante la fase di preparazione
- **manipolazione del risultato:** il risultato del comando, tradotto nelle strutture del linguaggio di programmazione, viene manipolato secondo la logica dell'applicazione



Esecuzione di comandi SQL

- Il risultato dell'esecuzione di un comando SQL è:
 - un insieme di tuple per le interrogazioni
 - un valore numerico per INSERT/UPDATE/DELETE e comandi del DDL



Esecuzione comandi SQL

- È necessario manipolare il risultato di un'interrogazione SQL (relazione) da linguaggio di programmazione
- Se l'interrogazione restituisce solo una tupla:
 - il numero di informazioni da analizzare è noto a tempo di compilazione e pari al numero degli attributi della relazione
 - è possibile definire variabili di comunicazione da utilizzare per inserire i valori degli attributi della tupla restituita
- Se l'interrogazione restituisce più tuple:

SE I "FILTRI"
INSERITI NELLE
SQL FILTRANO
TUTTE TRANNE
1

SE I FILTRI NE LASCIANO ≥ 2



Cursore


- Un cursore è un puntatore ad una tupla contenuta nel risultato di un'interrogazione SQL
- E' associato alla valutazione di un'interrogazione
- In ogni momento, la tupla puntata dal cursore, e quindi i valori degli attributi di tale tupla, possono essere letti ed inseriti in una struttura dati del programma

Cursore



Result Set

7369	SMITH	CLERK
7566	JONES	MANAGER
7788	SCOTT	ANALYST
7876	ADAMS	CLERK
7902	FORD	ANALYST

cursor  Current Row



Operazioni sui cursori

- **Dichiarazione**: associa un cursore ad un'interrogazione
- **Apertura**: esegue l'interrogazione associata al cursore e lo inizializza
- **Posizionamento**: sposta il cursore sulla tupla successiva o precedente rispetto a quella correntemente puntata (o sulla prima o sull'ultima tupla del risultato)
- **Chiusura**: disabilita il cursore

SQL statico

IL RISULTATO È UNO SOLO



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: vantaggio

- È nota a priori la struttura delle interrogazioni e dei risultati
- Rende possibile l'ottimizzazione delle istruzioni SQL durante la fase di compilazione dell'applicazione
- L'ottimizzatore del DBMS:
 - compila l'istruzione SQL
 - crea il piano di esecuzione
 - esecuzione più efficiente

SQL dinamico

PIU RISULTATI (UTILIZZO CURSORE)



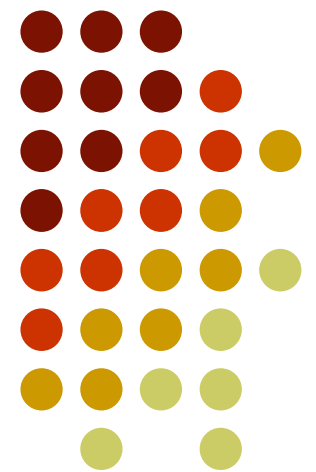
- Uso di comandi SQL noti solo a tempo di esecuzione
 - dipendono dal flusso di esecuzione
 - possono essere dati in input dall'utente
- Esempio: clausola WHERE in un comando SELECT che cambia (ad es. contiene o meno un determinato congiunto) in relazione al fatto che una certa condizione (che può dipendere da valori presi in input) sia vera o falsa



SQL dinamico

- I comandi SQL dinamici comportano maggiore flessibilità ma anche maggiori problemi in termini di ottimizzazione, compilazione ed esecuzione
- Un comando SQL statico può essere preparato durante la compilazione del programma ed il risultato della preparazione può essere utilizzato un numero arbitrario di volte, nel contesto di esecuzioni distinte
- Un comando SQL dinamico può invece essere preparato soltanto durante l'esecuzione del programma, con un conseguente aumento del tempo di risposta

Librerie di funzioni





Librerie di funzioni

- L'accesso alla base di dati avviene tramite un'interfaccia chiamata **Call Level Interface (CLI)**
- Ogni **CLI** è una libreria per uno specifico linguaggio
- La maggioranza dei DBMS offre soluzioni di questo tipo, che però sono proprietarie e quindi vanno bene solo per uno specifico DBMS
- Lo standard SQL prevede SQL/CLI
- Standard de facto: ODBC, JDBC



Librerie di funzioni

I comandi SQL sono inviati al DBMS per mezzo di funzioni del linguaggio ospite

- il programma ospite contiene direttamente le chiamate alle funzioni messe a disposizione dalla CLI
- le istruzioni SQL sono passate come parametri alle funzioni del linguaggio ospite



Modo d'uso

Indipendentemente dalla CLI adottata, esiste una strutturazione comune dell'interazione con il DBMS:

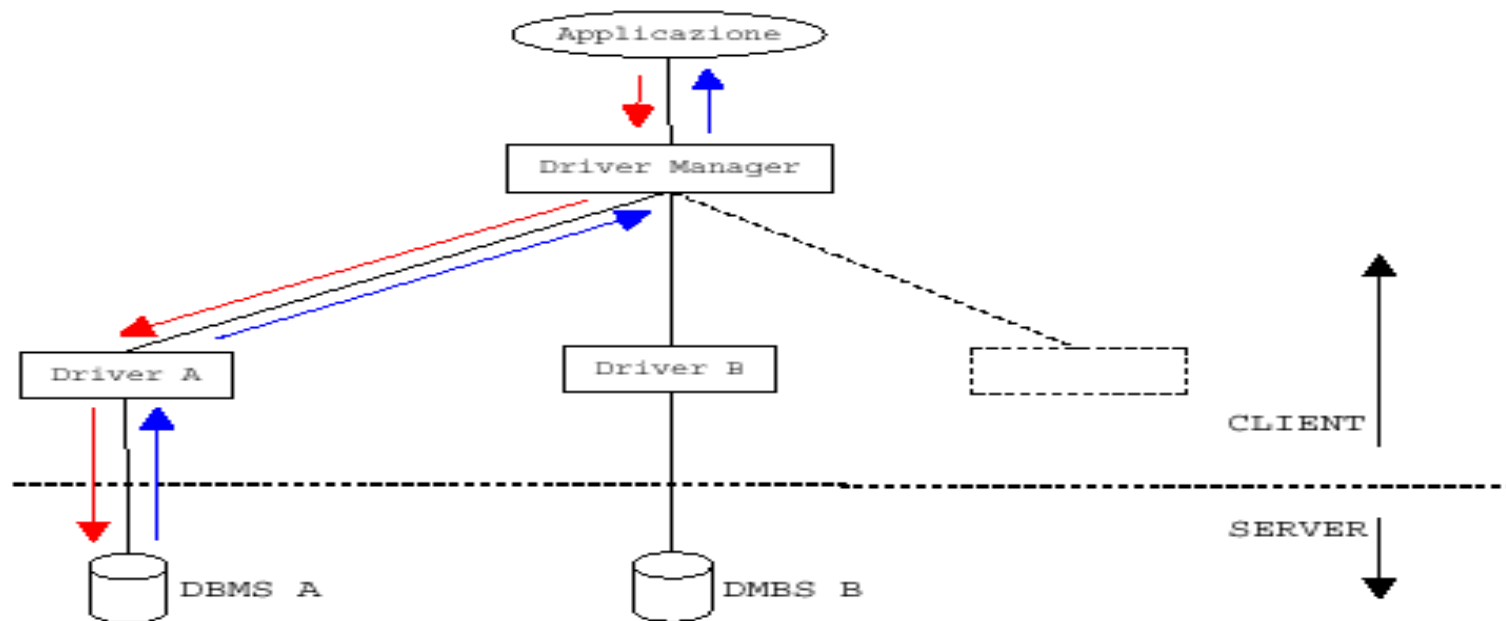
- apertura della connessione con il DBMS e DB
- esecuzione di istruzioni SQL
- chiusura della connessione

JDBC



- JDBC è una CLI Java standard per interagire con basi di dati da Java
- Cerca di essere il più semplice possibile rimanendo al contempo flessibile
- Permette di ottenere una soluzione “pure Java” per l’interazione con DBMS
 - indipendenza dalla piattaforma

JDBC: architettura di riferimento





DBMS

- Il DBMS sostanzialmente rimane inalterato nel suo funzionamento
- Riceve sempre e solo richieste nel linguaggio supportato
- Esegue il comando SQL ricevuto dal driver e invia i risultati all'applicazione



JDBC

- La portabilità a livello di eseguibile è ottenuta attraverso **driver**
- Un driver è un programma che traduce tutte le chiamate JDBC in chiamate specifiche per un DBMS



Driver Manager

- È una classe che gestisce la comunicazione tra applicazione e driver
- Risolve problematiche comuni a tutte le applicazioni:
 - quale driver caricare, basandosi sulle informazione fornite dall'applicazione
 - caricamento driver
 - chiamate alle funzioni dei driver
- L'applicazione interagisce solo con il driver manager



Applicazione

- Un'applicazione è un programma che chiama specifiche funzioni **JDBC** per accedere ai dati gestiti da un **DBMS**
- **Flusso tipico:**
 - caricamento driver
 - selezione sorgente dati (DBMS e specifico database) e connessione
 - invio statement SQL
 - recupero risultati e elaborazione
 - disconnessione

Classi JDBC

- DriverManager
- Connection
- Statement, con sotto-classe:
 - PreparedStatement:
 - CallableStatement
- ResultSet
- SQLException

FARE FOGLIO DOVE
PER OGNI CLASSE
HO IL METODO,
COSA FA, COME
DEVE ESSERE
COMPILATO





Caricamento driver

- Il primo passo in un'applicazione JDBC consiste nel caricare il driver che si intende utilizzare
- Ogni driver = classe Java
 - `Class.forName("oracle/jdbc.driver.OracleDriver");`
 - carica dinamicamente il driver per il DBMS Oracle
- Il nome della classe da usare viene fornito con la documentazione relativa al driver



Connessione

- La connessione avviene usando il metodo `getConnection` della classe `DriverManager`, che restituisce un oggetto di classe `Connection`:

```
Connection con =  
    DriverManager.getConnection("jdbc:oracle:www.bookstore.com:3083",  
                                userID, password);
```

- Una connessione può essere chiusa tramite il metodo `close` della classe `Connection`



JDBC - tipi di dato

- JDBC definisce un insieme di tipi SQL, che vengono poi mappati in tipi Java
- Gli identificatori sono definiti nella classe `java.sql.Types`

JDBC - tipi di dato



JDBC Types Mapped to Java Types	
JDBC Type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp



Elaborazione

- L'esecuzione di un'istruzione SQL richiede l'uso della classe Statement (o sotto-classi)
- Ogni oggetto Statement è associato a una connessione
- I comandi SQL possono essere:
 - generici -- non preparati (SQL statico e dinamico)
 - preparati (SQL statico)
 - callable statement (stored procedure)



Esecuzione comando SQL

- E' necessario creare un oggetto **Statement** associato a una **connessione** tramite il **metodo createStatement** della classe **Connection**:
Connection con;
...
Statement stmt = con.createStatement();
- L'oggetto statement non è ancora associato all'istruzione SQL da eseguire:
 - tale istruzione verrà specificata al momento dell'esecuzione



Esecuzione comandi SQL

- É necessario distinguere tra comandi che rappresentano query e comandi di aggiornamento

- Per eseguire una query:

```
stmt.executeQuery("SELECT * FROM Film");
```

- Per eseguire una operazione di aggiornamento, inclusi gli statement DDL:

```
stmt.executeUpdate("INSERT INTO Film VALUES( ....)");
```

- Per comandi di cui non è nota la tipologia:

```
execute
```

Esempio



```
stmt.executeUpdate("CREATE TABLE Film  
    titolo VARCHAR(30),  
    regista VARCHAR(20)  
    anno DECIMAL(4) NOT NULL,  
    genere CHAR(15) NOT NULL,  
    valutaz NUMERIC(3,2),  
    PRIMARY KEY(titolo,regista))");
```

```
stmt.executeUpdate("INSERT INTO Film  
    VALUES ('la tigre e la neve', 'roberto  
    benigni',2005,'commedia',3.25)");
```



Comandi preparati

- Un comando SQL “preparato” è:
 - compilato una sola volta
 - all’inizio dell’esecuzione dell’applicazione
 - eseguito più volte: QUERY CHE VA RIPETUTA UN PO DI VOLTE NEL CODICE
 - prima di ogni esecuzione è necessario specificare il valore corrente dei parametri
 - Modalità utile quando è necessario ripetere più volte l’esecuzione della stessa istruzione SQL
 - permette di ridurre il tempo di esecuzione
 - la compilazione è effettuata una volta sola



Creazione comandi preparati

- Sono oggetti istanze della classe `PreparedStatement`:

```
PreparedStatement queryPstmt =  
    con.prepareStatement("SELECT * FROM Film");
```

- La creazione di un comando preparato richiede la specifica della stringa corrispondente al comando SQL che dovrà poi essere eseguito



Esecuzione comandi preparati

- É necessario distinguere tra comandi che rappresentano query e comandi di aggiornamento
- Per eseguire una query:
`queryPstmt.executeQuery();`
- Per eseguire una operazione di aggiornamento, inclusi gli statement DDL:
`queryPstmt.executeUpdate();`



Uso di parametri

- È possibile specificare che la stringa che rappresenta il comando SQL verrà completata con parametri al momento dell'esecuzione

LA BASE È LA
QUERY
PREPARATA, POI
PER OGNI UTILIZZO
VA
EVENTUALMENTE
MODIFICATA

- I parametri sono identificati da '?':

```
PreparedStatement queryParPstmt = con.prepareStatement(  
    "SELECT * FROM Film WHERE genere = ?");
```

- I valori dei parametri vanno poi associati al comando, prima della sua esecuzione



Uso di parametri

- È possibile associare valori ai parametri usando il metodo `setXXX`, dove XXX indica un tipo Java:
 - `setString`, per stringhe
 - `setInt`, per interi
 - `setBigDecimal`, per number
- Esempio:
 - `queryParPstmt.setString(1, 'thriller');`
 - `queryParPstmt.executeQuery();`

QUA HO LA QUERY DI
BASE CHE PRENDE IN
INPUT VALORI, IO
SPECIFICO IL TIPO DEI
VALORI E DENTRO LA
PARENTESI COSA
INSERISCO



Cursori

- **JDBC** restituisce i risultati di una query in un oggetto istanza della classe **ResultSet**
- E' possibile accedere a ciascuna tupla risultato della query mediante metodi della classe **ResultSet**
- Il metodo **next()** permette di spostarsi nel result set (cursore):
 - `while (rs.next()) { /* accedi alla tupla corrente */ }`
- Inizialmente il cursore è posizionato prima della prima tupla
- Il metodo diventa falso quando non ci sono più tuple da analizzare



Elaborazione risultato

- I metodi `getXXX`, della classe `ResultSet`, permettono di recuperare il valore associato ad un certo attributo, puntato correntemente dal cursore
- XXX è il tipo Java nel quale il valore deve essere convertito:

```
String s = rs.getString("Cognome");
```

- Gli attributi possono anche essere acceduti tramite la notazione posizionale:
 - `String s = rs.getString(2);`
 - `int n = rs.getInt(5);`

Esempio

QUA POTREI METTERE
DELLE ALTRE QUESRY PER
FILTRARE ALTRI ELEMENTI

DICHIARAZIONE DI UNA QUERY
PREPARATA

CREO UNA STRINGA
PERCHE QUELLO CHE
MI STAMPERA SARA
UNA STRINGA, FOSSE
INTERO ALLORA INT
ECC E METTO DENTRO
COSA DEVE CERCARE

```
Statement selStmt = con.createStatement ();  
String stmt = "SELECT titolo, regista  
FROM Film  
WHERE genere = 'thriller';"  
ResultSet rsStmt = selStmt.executeQuery (stmt);
```

DICHIARAZIONE
DEL RESULT
ATO = QUERY
GENEALE.EXEC
UTE (NOME
DELLA
STRINGA)

PIU RISULTATI POSSIBILI
QUINDI .NEXT NELLA QUERY RESULT

RICHIAMO QUERY
GENEALE E USO
IL . PER
CHIAMARARE IL
METODO ESEGUI
CON IO NOME
DELLA STRINGA

```
while (rsStmt.next())  
{  
    System.out.println (rsStmt.getString ("titolo"));  
    System.out.println (rsStmt.getString ("regista"));  
}
```

STAMPA QUELLO CHE HO SELEZIONATO

CICLO PER SELEZIONARE LE POSSIBILITA FILTRATE



Disconnessione

- Per risparmiare risorse, è utile chiudere gli oggetti di classe Connection, Statement, ResultSet quando non vengono più utilizzati
 - Metodo close()
- La chiusura di un oggetto di tipo Connection chiude tutti gli Statement associati mentre la chiusura di uno Statement chiude i ResultSet associati

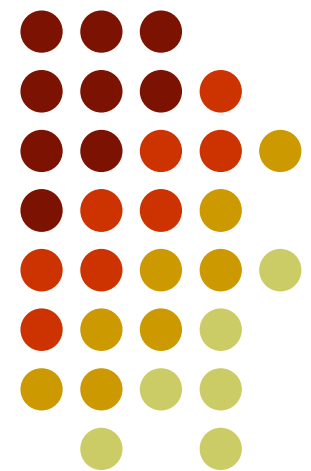


Eccezioni

- La classe `java.sql.SQLException` estende la classe `java.lang.Exception` in modo da fornire informazioni ulteriori in caso di errore di accesso alla base di dati, tra cui:
 - Il codice di errore specifico al DBMS:
 - `getErrorCode()`
 - Una descrizione dell'errore:
 - `getMessage()`

SQL Ospitato

Embedded SQL





Concetti di base

- I comandi SQL vengono direttamente inseriti in programmi scritti in un'estensione di un linguaggio di programmazione, chiamato **linguaggio ospite** (C, C++, Java)
- Il linguaggio ospite permette di utilizzare SQL senza il bisogno di funzioni di interfaccia
- I comandi SQL (embedded SQL) possono apparire in ogni punto in cui può comparire un'istruzione del linguaggio ospite

SLIDE PRIMA
SONO
SCRITTE IN
JAVA CHE È IL
LINGUAGGIO
OSPITE.
INSERENDO
LE QUERY
DENTRO A
JAVA



Concetti di base

- Ogni istruzione SQL deve essere chiaramente identificabile nel testo del programma:
 - preceduta da un prefisso e seguita da un terminatore
- Per molti linguaggi (C, Pascal, Fortran, Cobol):
 - prefisso = EXEC SQL
 - terminatore = ;



Architettura di riferimento

- L'esecuzione dei programmi avviene in tre passi: precompilazione, compilazione, esecuzione
 - **precompilazione:**
 - Il programma viene passato ad un precompilatore specifico della combinazione linguaggio-DBMS-sistema operativo
 - Il precompilatore elimina i comandi SQL sostituendoli con chiamate al DBMS espresse nella sintassi del linguaggio ospite
 - Il risultato è un programma scritto completamente nel linguaggio ospite
 - **compilazione:**
 - Il programma risultato della fase precedente viene compilato utilizzando un compilatore per il linguaggio ospite



Esempio

```
include<stdlib.h>
```

```
main()
```

```
{
```

```
exec sql begin declare section;
```

```
char NomeDip = "Manutenzione";
```

```
char CittaDip = "Pisa";
```

```
int NumeroDip = 20;
```

```
exec sql end declare section;
```

DEFINIZIONE DATI

```
exec sql connect to utente@librobd;
```

```
if (sqlca.sqlcode != 0) {
```

```
    printf("Connessione al DB non riuscita\n"); }
```

```
else {
```

```
exec sql insert into Dipartimento values(:NomeDip,:CittaDip,:NumeroDip);
```

```
exec sql disconnect all;
```

```
}
```

```
}
```

ISTRUZIONI SQL

ISTRUZIONI SQL



Embedded SQL

- EXEC SQL denota le porzioni di interesse del precompilatore:
 - definizioni dei dati
 - istruzioni SQL
- Le variabili del programma possono essere usate come “parametri” nelle istruzioni SQL (precedute da “:”) dove sintatticamente sono ammesse costanti

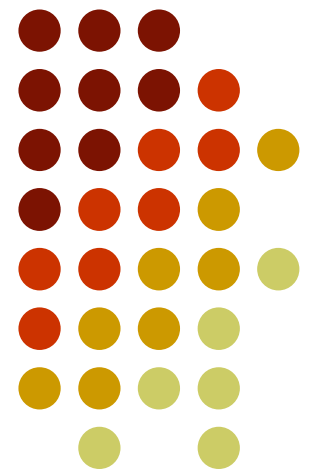


Embedded SQL

- sqlca è una struttura dati per la comunicazione fra programma e DBMS
- sqlcode è un campo di sqlca che mantiene il codice di errore dell'ultimo comando SQL eseguito:
 - zero: successo
 - altro valore: errore o anomalia

Estensioni procedurali: Oracle PL /SQL

loop while if scorrimento puntatori





Stored procedure

Una stored procedure è una funzione o una procedura definita all'interno del DBMS

- fa parte dello schema logico della base di dati
- 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

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



Esempi successivi

Gli esempi nel seguito si riferiscono, nella maggioranza dei casi, alle seguenti tabelle:

```
CREATE TABLE Dipartimenti  
(nome VARCHAR(24) PRIMARY KEY,  
sede VARCHAR(30),  
budget NUMERIC(7,2));
```

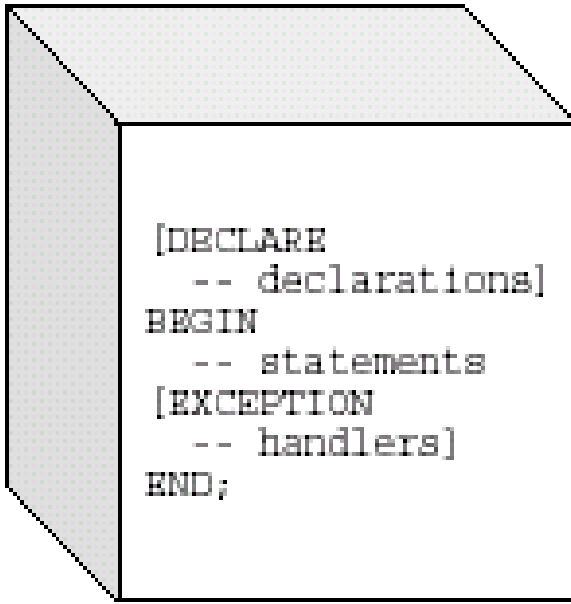
```
CREATE TABLE Impiegati  
(nome VARCHAR(24) PRIMARY KEY,  
indirizzo VARCHAR(30),  
dipartimento VARCHAR(24) REFERENCES Dipartimenti,  
stipendio NUMERIC(6,2));
```

6,2 INDICA CHE LO STIPENDIO AVRÀ 6 CIFRE E DUE DECIMALI
ESEMPIO 50000,43



PL/SQL

- È un linguaggio con struttura a blocchi
- Ogni blocco è composto da tre parti:
 - parte dichiarativa
 - parte di esecuzione
 - parte di gestione delle eccezioni
- Nel seguito vedremo solo alcuni aspetti base di PL/SQL



```
[DECLARE  
  -- declarations]  
BEGIN  
  -- statements  
[EXCEPTION  
  -- handlers]  
END;
```



Stored procedure in Oracle

- Creazione di una stored procedure in Oracle:

```
CREATE [OR REPLACE] PROCEDURE  
<nomeStoredProcedure> [(elencoParametri)] IS  
({istruzioneSQL|bloccoPL/SQL});
```

- La stored procedure può essere associata a:
 - una singola istruzione SQL
 - un blocco di codice scritto in PL/SQL

Blocco PL/SQL - dichiarazioni



- Le dichiarazioni devono essere precedute dalla parola chiave **DECLARE**
- È possibile dichiarare sia costanti che variabili ed usarle ovunque possa comparire un'espressione
- Le variabili possono avere un qualunque tipo SQL
- È possibile assegnare un valore di default alle variabili al momento della dichiarazione

Dichiarazioni - Esempio



DECLARE

data_nascita DATE;

num_impiegati INTEGER := 0;

manager BOOLEAN;

bonus NUMERIC(4,2);



Uso di attributi in dichiarazioni

Possibilità di specificare che il tipo di una variabile coincide con il tipo di un'altra variabile o di un attributo di una tabella

- `imp impiegati.nome%TYPE;`

- il tipo della variabile `imp` coincide con il tipo dell'attributo `nome` della tabella `impiegati`

- `credito NUMERIC(7,2);`
`debito credito%TYPE;`

- il tipo della variabile `debito` coincide con il tipo della variabile `credito`

DEVBITO E CREDITO
SONO DELLO STESSO
RTIPO numeric PERCHE
SONO DUE VALORI CON
LA VIRGOLA, QUINDI PER
EVITARE DI SCRIVERE
DUE VOLTE NUMERIC E LA
LUNGHEZZA FACCIO5TYPE



Uso di attributi in dichiarazioni

Possibilità di specificare che una variabile rappresenta un record, corrispondente ad una tupla di una tabella:

- `dip_rec dipartimenti%ROWTYPE;`
 - la variabile `dip_rec` è dichiarata come un record, i cui elementi hanno gli stessi nomi e tipi degli attributi della tabella `dipartimenti`

IL RECORD HA GLI STESSI TIPI DELLA TABELLA E GLI STESSI NOMI



Strutture di controllo

Costrutti di scelta:

- IF THEN, IF THEN ELSE, IF THEN ELSIF
- come in C

```
IF condition1 THEN
sequence_of_statements1
ELSIF condition2 THEN
sequence_of_statements2
ELSE
sequence_of_statements3
END IF;
```



Strutture di controllo

Cicli:

- **LOOP:** LOOP
sequence_of_statements
EXIT WHEN boolean_expression;
END LOOP;
- **WHILE LOOP:** WHILE condition LOOP
sequence_of_statements
END LOOP;
- **FOR LOOP:** FOR counter IN [REVERSE]
lower_bound..higher_bound
LOOP
sequence_of_statements
END LOOP;



Strutture di controllo

GOTO:

- è necessario associare label a determinati punti del programma
- con GOTO è possibile modificare il flusso, portandolo ad eseguire l'istruzione associata alla label specificata

BEGIN

...

GOTO insert_row;

...

<<insert_row>>

INSERT INTO employee VALUES ...

End;



Istruzioni - Esempio

```
DECLARE
nome_i impiegati.nome%TYPE;
stipendio_i impiegati.stipendio%TYPE;
....
BEGIN
...
SELECT nome, stipendio
INTO nome_i, stipendio_i FROM impiegati
WHERE nome = 'Rossi';
...
END;
```



Istruzioni - Esempio

```
BEGIN
```

```
...
```

```
IF stipendio > 50000 THEN
```

```
  bonus := 1500;
```

```
ELSIF stipendio > 35000 THEN
```

```
  bonus := 500;
```

```
ELSE
```

```
  bonus := 100;
```

```
END IF;
```

```
INSERT INTO stipendi VALUES (id_imp, bonus, ...);
```

```
END;
```

Cursori



- Dichiarazione:
 - **CURSOR** <nome_cursore> [(parametro[, parametro]...)]
IS select_statement;
- Apertura:
 - **OPEN** <nome_cursore>;
- Avanzamento:
 - **FETCH** <nome_cursore> **INTO** <nome_record>
 - **FETCH** <nome_cursore > **INTO** < lista_variabili >
 - Il cursore viene associato ad una variabile booleana **NOT FOUND** che diventa vera quando non ci sono più tuple da analizzare
- Chiusura:
 - **CLOSE** <nome_cursore>

Esempio



```
DECLARE  
emp_rec impiegati%ROWTYPE;  
mio_dip impiegati.dipartimento%TYPE;  
CURSOR c1 IS SELECT * FROM impiegati  
WHERE dipartimento = mio_dip;
```

definizione

metto la query
che mi stampa
piu risultati e ci
dichiaro il
puntatore

```
BEGIN
```

```
...
```

```
OPEN c1;
```

apertura

```
LOOP
```

```
    FETCH c1 INTO emp_rec;
```

avanzamento

```
    EXIT WHEN c1%NOTFOUND;
```

```
....
```

```
END LOOP;
```

```
CLOSE c1;
```

chiusura

```
END;
```



Approcci all'integrazione

Le tecniche discusse 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



Conclusioni: JDBC

- Vantaggi:
 - portabilità
 - utilizzo di Java come unico paradigma
- Svantaggi:
 - performance
 - minor facilità di programmazione rispetto alle altre soluzioni



Conclusioni: Embedded SQL

- Vantaggi:
 - portabilità (modulo la precompilazione)
 - facilità di programmazione rispetto ad utilizzo di API
- Svantaggi:
 - performance
 - minor facilità di programmazione rispetto alle estensioni procedurali di SQL

Conclusioni: Estensioni procedurali di SQL



- Vantaggi:
 - facilità di programmazione
 - performance
 - gestione ottimizzata delle eccezioni del DBMS
 - tool a supporto della programmazione forniti dal DBMS
- Svantaggi:
 - espressività
 - portabilità