Interoperabilità nei sistemi di gestione dei dati

1

Sommario

- Introduzione
- Architettura di riferimento per ODBC,JDBC
- Flusso applicazioni
- ODBC
  - architettura
  - livelli di conformità
  - il flusso generale di un'applicazione ODBC
  - le primitive attraverso un esempio
  - ODBC in Oracle
- JDBC
  - architettura
  - tipologie di driver
  - le classi principali
  - il flusso generale di un'applicazione ODBC
  - i metodi attraverso un esempio
  - JDBC in Oracle

2

Introduzione

- Il problema principale nelle architetture client-server è l'esistenza di client eterogenei:
  - Backend: database server
  - Frontend: gui/forms/interfaccie web
- Nasce il problema di capire come le applicazioni client possano comunicare con il server:
  - come descrivere le query
  - come descrivere i risultati delle query
- Prima soluzione: uso SQL
  - Svantaggi: cambiando server potrebbe essere necessario cambiare il client, in quanto lo specifico dialetto SQL potrebbe cambiare
- Necessità di accedere i dati in modo interoperabile, cioè indipendente dallo specifico server considerato
  - se cambia il server non cambia l'applicazione
  - i tempi di sviluppo applicazioni client si riducono

Introduzione

- L'interoperabilità rappresenta il problema principale nello sviluppo di applicazioni eterogenee per sistemi distribuiti
- Richiede la disponibilità di funzioni di adattabilità e di conversione che rendano possibile lo scambio di informazione tra sistemi, reti ed applicazioni, anche se eterogenei
- l'interoperabilità è resa possibile dall'utilizzo di protocolli standard come FTP, SMTP/MIME, ecc.
- In riferimento ai sistemi di gestione dei dati, l'interoperabilità ha richiesto lo sviluppo di standard adeguati, nella forma di API (Application Program Interface)
  - ODBC
  - JDBC

Introduzione

- Per comprendere la necessità di comunicare attraverso API con un DBMS, consideriamo come sia possibile in generale realizzare questa comunicazione:
  - embedded SQL
  - moduli SQL
  - call-level interface (CLI)

5

Introduzione

- Embedded SQL:
  - SQL "inserito" in un linguaggio di programmazione
  - statement processati da uno speciale precompilatore
  - visto in DB1
  - può essere:
    - statico (statement noti a compile-time)
    - dinamico (statement generati a run-time)
  - meccanismo di interazione standard ma il codice dipende dal DBMS prescelto
    - cambiando DBMS l'applicazione deve essere nuovamente compilata
  - Esempio: per Java SQLJ

6

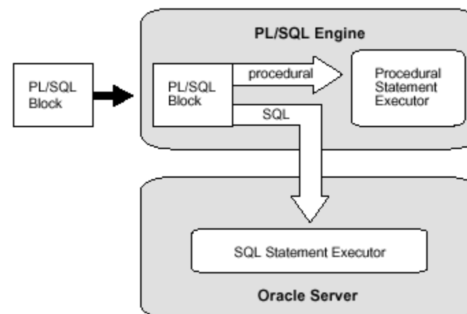
Introduzione

- Moduli SQL
  - un modulo consiste di un gruppo di procedure, chiamate da un linguaggio di programmazione ospite, ogni procedura contiene un singolo statement SQL
  - un modulo può essere interpretato come un oggetto di libreria legato al codice applicativo
  - questo collegamento dipende dall'implementazione:
    - le procedure possono essere compilate e linkate al codice applicativo
    - possono essere compilate e memorizzate nel DBMS e chiamate dal codice applicativo
    - possono essere interpretate
  - chiara separazione tra statement SQL e linguaggio di programmazione
  - meccanismo di interazione standard ma il codice dipende dal DBMS prescelto
    - cambiando DBMS l'applicazione deve essere nuovamente compilata

7

Introduzione

- Esempio:
  - in Oracle, moduli scritti in PL/SQL
  - compilati, memorizzati nel DBMS



8

Introduzione

- Call-level interface
  - libreria di funzioni del DBMS che possono essere chiamate dai programmi applicativi
  - simile alle tipiche librerie C
  - passi tipici:
    - l'applicazione effettua una chiamata ad una funzione CLI per connettersi al DBMS
    - l'applicazione crea uno statement SQL e lo inserisce in un buffer, quindi chiama una o più funzioni CLI per inviare lo statement al DBMS per l'esecuzione
    - se lo statement è di tipo SELECT, la funzione restituisce le tuple del risultato
    - se è di tipo INSERT, DELETE, UPDATE, la funzione restituisce il numero di tuple modificate
    - se è di tipo DDL, non viene restituito alcun valore significativo
    - l'applicazione effettua una chiamata ad una funzione CLI per disconnettersi dal DBMS

9

Introduzione

- Esempio: la CLI di Oracle si chiama OCI (Oracle Call Interface)
- Esistono alcuni standard per CLI:
  - X/Open Specification SQL CLI
  - ISO/IEC SQL/CLI

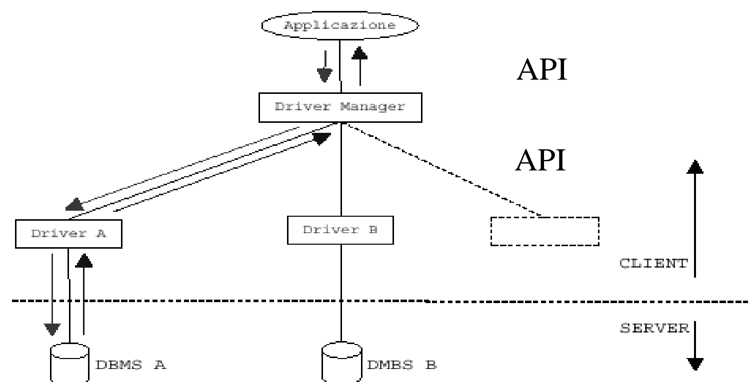
10

Introduzione

- Tra i tre livelli di interazione precedenti, le CLI sembrano il meccanismo più adeguato per garantire interoperabilità:
  - indipendenti dal codice client
  - l'applicazione cambia interazione con il server chiamando in modo opportuno le funzioni di libreria
  - chiara distinzione tra interfaccia ed implementazione
  - una stessa applicazione binaria può funzionare considerando diversi DBMS
- Necessità di standardizzazione
  - le applicazioni devono potere accedere DBMS diversi usando lo stesso codice sorgente
  - le applicazioni devono potere accedere DBMS diversi simultaneamente
  - ODBC, JDBC si possono vedere come CLI ottenute come risultato del processo di standardardizzazione
    - ODBC: libreria C
    - JDBC: libreria Java

11

Architettura di riferimento



12

Applicazione

- Un'applicazione è un programma che chiama specifiche funzioni API per accedere ai dati gestiti da un DBMS
- Flusso tipico:
  - selezione sorgente dati (DBMS e specifico database) e connessione
  - sottomissione statement SQL per l'esecuzione
  - recupero risultati e processamento errori
  - commit o rollback della transazione che include lo statement SQL
  - disconnessione

13

Driver Manager

- È una libreria 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

14

Driver

- Sono librerie dinamicamente connesse alle applicazioni che implementano le funzioni API
- ciascuna libreria è specifica per un particolare DBMS
  - driver Oracle è diverso dal driver Informix
- traducono le varie funzioni API nel dialetto SQL utilizzato dal DBMS considerato (o nell'API supportata dal DBMS)
- il driver maschera le differenze di interazione dovute al DBMS usato, il sistema operativo e il protocollo di rete
- Si occupano in particolare di:
  - iniziare transazioni
  - sottomettere statement SQL
  - inviano dati e recuperano dati
  - gestiscono errori

15

DBMS

- Il DBMS sostanzialmente rimane inalterato nel suo funzionamento
- riceve sempre e solo richieste nel linguaggio supportato
- esegue lo statement SQL ricevuto dal driver e invia i risultati

16

Esempio

- Nel seguito gli esempi si riferiranno alla tabella Impiegati definita in SQL come segue:

```
CREATE TABLE Impiegati
(Matr VARCHAR(4),
Nome VARCHAR(20),
Cognome VARCHAR(20),
Manager VARCHAR(4),
Stipendio NUMBER);
```

17

ODBC (Open DataBase Connectivity)

- Standard proposto da Microsoft nel 1991
- Supportata da praticamente tutti i sistemi di gestione dati relazionali
- Offre all'applicazione un'interfaccia che consente l'accesso ad una base di dati non preoccupandosi di:
  - il particolare dialetto di SQL
  - il protocollo di comunicazione da usare con il DBMS
  - la posizione del DBMS (locale o remoto)
- è possibile connettersi ad un particolare DB tramite una DSN (Data Source Name), che contiene tutti i parametri necessari alla connessione con il DB:
  - protocollo di comunicazione
  - tipo di sorgente dati (es: Oracle DBMS)
  - specifico database

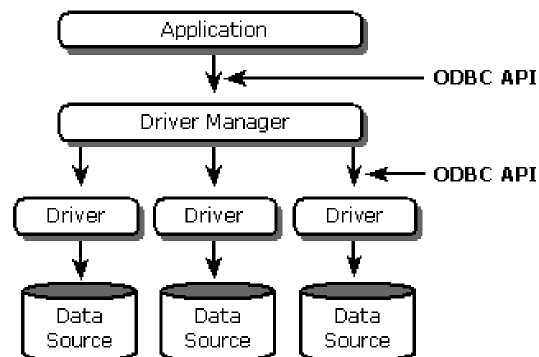
18

ODBC

- Il linguaggio supportato da ODBC è un SQL ristretto, caratterizzato da un insieme minimale di istruzioni
- Questa è una scelta obbligata se si vuole permettere un cambio di DBMS lasciando inalterati i client
- Il linguaggio permette di eseguire query statiche per applicazioni di tipo tradizionale o dinamiche, per applicazioni interattive
- Nel primo caso eventuali errori di SQL sono riportati al momento stesso della compilazione
- Architettura conforme a quando descritto in generale
- Funzioni implementate in C, quindi non completamente portabili
- è compatibile con X/open SQL CLI e ISO SQL/CLI

19

Architettura



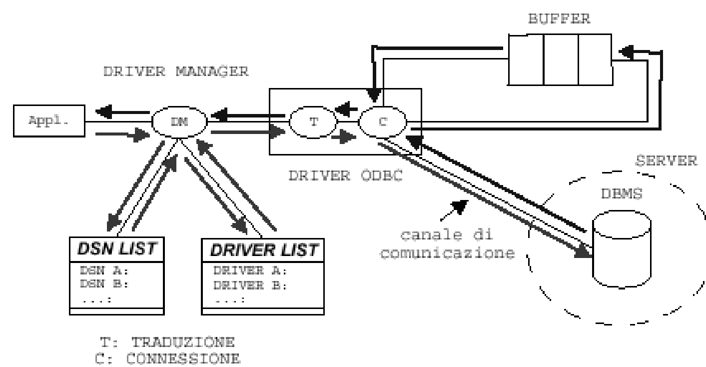
20

Architettura

- Driver Manager:
  - su Windows, fornito con il sistema operativo
  - con Linux, deve essere installato
- Driver:
  - forniti con il DBMS a cui si riferiscono
  - spesso installati e registrati a livello Driver Manager quando si installa il DBMS client

21

Funzionamento di una query ODBC



22

Come registrare una DSN?

- I driver ODBC mettono in genere a disposizione tool di amministrazione che permettono di specificare tutti i dettagli relativi ad un DSN (Data Source Name)
- DSN: Data Source Name, è la stringa che identifica:
  - tipo di driver
  - tipo di comunicazione
  - tipo di database
  - nome database
- Dopo la registrazione, queste informazioni sono a disposizione del Driver Manager

23

Vantaggi e svantaggi

- Vantaggi
  - astrazione (almeno teorica) rispetto al DBMS utilizzato
  - diffusione: supportato almeno parzialmente da molti DBMS commerciali e non
  - non modifica il server
- Svantaggi
  - per ogni coppia (piattaforma client, server) serve un driver specifico (strategia commerciale Microsoft)
  - gran parte del lavoro è demandata al driver, che diventa quindi cruciale per il funzionamento

24

Livelli di conformità

- DBMS diversi possono avere diverso potere espressivo
- Inoltre i driver possono implementare solo una parte della libreria ODBC
- I livelli di conformità permettono di specificare:
  - quali funzioni ODBC sono implementate dal driver
    - (livello di conformità dell'interfaccia)
  - quali statement SQL il DBMS è in grado di eseguire
    - (livello di conformità SQL)

25

Livelli di conformità di interfaccia

- Ogni livello di conformità è rappresentato da un insieme di funzioni API, per ciascuna delle quali viene specificata la modalità di chiamata (vincoli sui parametri)
- L'applicazione può recuperare il livello di conformità del driver invocando una opportuna funzione ODBC (`SQLGetInfo`)
- Sono stati definiti tre livelli:
  - Core: funzionalità di base
  - Livello 1: Core + funzionalità addizionali, supportate da tipici sistemi relazionali, come transazioni
  - Livello 2: Livello 1 + funzionalità di amministrazione, tra cui gestione cataloghi

26

Livelli di conformità SQL

- Ogni driver deve supportare una grammatica SQL-92 minima
- Esistono altri livelli, che permettono di supportare un insieme maggiore di statement SQL
  - Entry level
  - intermediate level
  - full level
- Possibilità di recuperare il livello con una opportuna funzione (`SQLGetInfo`)

27

Tipi di dato

- ODBC utilizza due insiemi di tipi di dato:
  - tipi di dato SQL, utilizzati nel database
  - tipi di dato C, da utilizzare nell'applicazione
- Tipi di dato SQL:
  - ogni database definisce i propri tipi di dato mentre ODBC definisce identificatori di tipi di dato
  - il driver poi stabilisce come questi tipi di dato debbano essere mappati sugli identificatori ODBC
  - Esempio: SQL_CHAR identificatore per CHAR
- Tipi di dato C:
  - ODBC definisce identificatori per i tipi di dato C (SQL_C_CHAR)
  - ODBC definisce anche un mapping da tipi di dato SQL a tipi di dato C

28

Alcuni identificatori di tipi di dato SQL

SQL type identifier	Typical SQL data type	Typical type description
SQL_CHAR	CHAR(*n*)	Character string of fixed string length *n*.
SQL_VARCHAR	VARCHAR(*n*)	Variable-length character string with a maximum string length *n*.
SQL_DECIMAL	DECIMAL(*p*,*s*)	Signed, exact, numeric value with a precision of at least *p* and scale *s*. (The maximum precision is driver-defined.) (1 <= *p* <= 15; *s* <= *p*).[4]
SQL_NUMERIC	NUMERIC(*p*,*s*)	Signed, exact, numeric value with a precision *p* and scale *s* (1 <= *p* <= 15; *s* <= *p*).[4]
SQL_SMALLINT	SMALLINT	Exact numeric value with precision 5 and scale 0 (signed: –32,768 <= *n* <= 32,767, unsigned: 0 <= *n* <= 65,535)[3].
SQL_INTEGER	INTEGER	Exact numeric value with precision 10 and scale 0 (signed: –2[31] <= *n* <= 2[31] – 1, unsigned: 0 <= *n* <= 2[32] – 1)[3].
SQL_REAL	REAL	Signed, approximate, numeric value with a binary precision 24 (zero or absolute value 10[–38] to 10[38]).

29

Alcuni identificatori di tipi di dato SQL

SQL type identifier	Typical SQL data type	Typical type description
SQL_FLOAT	FLOAT(*p*)	Signed, approximate, numeric value with a binary precision of at least *p*. (The maximum precision is driver-defined.)[5]
SQL_DOUBLE	DOUBLE PRECISION	Signed, approximate, numeric value with a binary precision 53 (zero or absolute value 10[–308] to 10[308]).
SQL_BIT	BIT	Single bit binary data.[8]
SQL_BIGINT	BIGINT	Exact numeric value with precision 19 (if signed) or 20 (if unsigned) and scale 0 (signed: –2[63] <= *n* <= 2[63] – 1, unsigned: 0 <= *n* <= 2[64] – 1)[3],[9].
SQL_BINARY	BINARY(*n*)	Binary data of fixed length *n*.[9]
SQL_TYPE_DATE[6]	DATE	Year, month, and day fields, conforming to the rules of the Gregorian calendar
SQL_TYPE_TIME[6]	TIME(*p*)	Hour, minute, and second fields, with valid values for hours of 00 to 23, valid values for minutes of 00 to 59, and valid values for seconds of 00 to 61. Precision *p* indicates the seconds precision.
SQL_TYPE_TIMESTAMP[6]	TIMESTAMP(*p*)	Year, month, day, hour, minute, and second fields, with valid values as defined for the DATE and TIME data types.

30

Alcuni identificatori di tipi di dato C

C type identifier	ODBC C typedef	C type
SQL_C_CHAR	SQLCHAR *	unsigned char *
SQL_C_SSHORT	SQLSMALLINT	short int
SQL_C_USHORT	SQLUSMALLINT	unsigned short int
SQL_C_SLONG	SQLINTEGER	long int
SQL_C_ULONG	SQLUINTEGER	unsigned long int
SQL_C_FLOAT	SQLREAL	float
SQL_C_DOUBLE	SQLDOUBLE, SQLFLOAT	double
SQL_C_BIT	SQLCHAR	unsigned char
SQL_C_BINARY	SQLCHAR *	unsigned char *

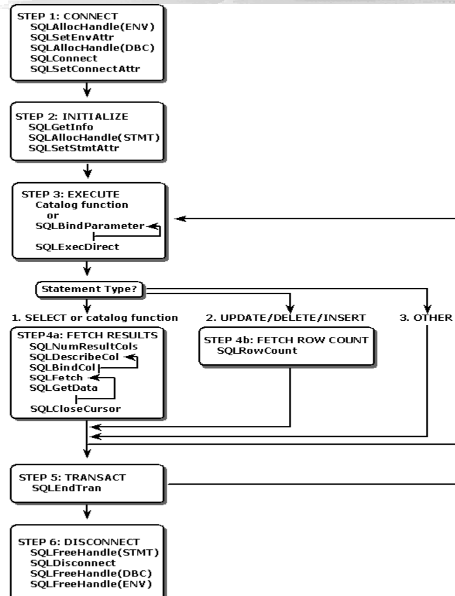
31

Alcuni identificatori di tipi di dato C

C type identifier	ODBC C typedef	C type
SQL_C_TYPE_DATE[c]	SQL_DATE_STRUCT	struct tagDATE_STRUCT {
		SQLSMALLINT year;
		SQLSMALLINT month;
		SQLSMALLINT day;
		} DATE_STRUCT;
SQL_C_TYPE_TIME	SQL_TIME_STRUCT	struct tagTIME_STRUCT {
		SQLSMALLINT hour;
		SQLSMALLINT minute;
		SQLSMALLINT second;
		} TIME_STRUCT;
SQL_C_TYPE_TIMESTAMP	SQL_TIMESTAMP_STRUCT	struct tagTIMESTAMP_STRUCT {
		SQLSMALLINT year;
		SQLSMALLINT month;
		SQLSMALLINT day;
		SQLSMALLINT hour;
		SQLSMALLINT minute;
		SQLSMALLINT second;
		} TIMESTAMP_STRUCT;

32

Flusso applicazione ODBC (3.51)



33

Flusso generale applicazione ODBC

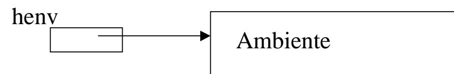
- Nel seguito vedremo solo le operazioni principali
- Ogni funzione che vedremo restituisce un valore di tipo `SQLRETURN`, che può valere:
  - `SQL_SUCCESS`, se l'operazione è andata a buon fine
  - `SQL_ERROR`, se si è verificato qualche errore
  - `SQL_SUCCESS_WITH_INFO`, una sorta di warning
- Nel seguito indicheremo solo i valori di ritorno diversi dai precedenti

34

Passo 1: Connessione

- Caricamento Driver Manager
  - L'esecuzione di questo passo dipende dal sistema operativo
- Allocazione **ambiente:**
  - buffer nel quale verranno registrate tutte le informazioni relative al driver e al DBMS verso il quale viene effettuata la connessione

```
SQLHENV henv1;
SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv1);
```



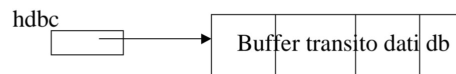
- Possibilità di settare parametri d'ambiente (funzione `SQLSetEnvAttr`, che non vediamo)
  - versione ODBC utilizzata

35

Connessione

- Definizione buffer di connessione: generazione buffer, atto a contenere informazioni relative alla connessione

```
SQLHDBC hdbc1;
SQLHENV henv1;
SQLAllocHandle(SQL_HANDLE_DBC,henv1,&hdbc1);
```



- Possibilità di settare parametri di connessione (funzione `SQLSetConnectAttr`, che non vediamo)
  - informazioni relative alla gestione di transazioni
- Il driver non viene ancora caricato
  - verrà caricato solo al momento della connessione

36

Connessione

- Connessione al database: tre possibilità, vediamo la più semplice:

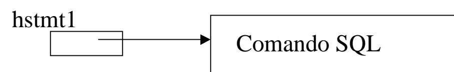
```
SQLHDBC hdbc1;
SQLConnect(hdbc,"DSN", SQL_NTS,
"id_utente", SQL_NTS, "password", SQL_NTS);

/* SQL_NTS = lunghezza del parametro
```

37

Passo 2: Inizializzazione

- Allocazione buffer per statement SQL

```
SQLHDBC hdbc1;
SQLHSTMT hstmt1;
SQLAllocHandle(SQL_HANDLE_STMT, hdbc1,&hstmt1);
```



- Possibilità di settare parametri di statement (funzione `SQLSetStmtAttr`, che non vediamo)
  - informazioni relative alla gestione del cursore

38

Esempio

```
SQLHENV      henv;        /* variabile ambiente
SQLHDBC      hdbc;        /* variabile buffer transito dati
SQLHSTMT     hstmt;       /* variabile comando SQL
SQLRETURN    re;          /* valore di ritorno

re = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,  &henv);

if(re == SQL_SUCCESS) {
   re = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
    if(re == SQL_SUCCESS) {
      re = SQLConnect(hdbc,
               ″ContiCorrenti″, SQL_NTS,    /*Riferimento al server
               ″Pippo″, SQL_NTS,            /*Utente
               ″password″, SQL_NTS);         /*Password dell'utente
       if(re == SQL_SUCCESS) {
         re = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);


         if(re == SQL_SUCCESS) {
             ″″″..  Interrogazioni ed elaborazione dati ″″″.
         }}}}
```

39

Passo 3: Costruzione ed esecuzione statement SQL

- Due modi principali di esecuzione:
  - *diretta*: l'applicazione definisce lo statement SQL (eventualmente preso come input), che viene ottimizzato ed eseguito in un unico passo a run-time
    - utile quando lo statement viene utilizzato una sola volta
  - *preparata*: l'applicazione definisce lo statement SQL che viene ottimizzato ed eseguito a run-time in due passi distinti
    - il piano di accesso viene generato e mantenuto per le successive esecuzioni
    - utile quando lo statement deve essere eseguito più volte
    - spesso si utilizzano parametri da settare dopo la preparazione e prima dell'esecuzione (non lo vediamo)

40

Costruzione ed esecuzione di statement SQL

- Esecuzione diretta:

```
SQLHSTMT hstmt1;
SQLCHAR* SQLStatement;
SQLExectDirect(hstmt1,SQLStatement, SQL_NTS);
```

- Esecuzione preparata:

```
SQLHSTMT hstmt1;
SQLCHAR* SQLStatement;
SQLPrepare(hstmt1,SQLStatement, SQL_NTS);
SQLExecute(hstmt1);
```

41

Esempio esecuzione

Diretta:

```
SQLExectDirect(hstmt, ”SELECT* FROM Impiegati”);
```

Preparata:

```
SQLPrepare(hstmt,”SELECT*FROM Impiegati”,SQL_NTS);
SQLExecute(hstmt);
```

Per ogni successiva esecuzione non devo più invocare SQLPrepare

42

Passo 4: Elaborazione risultato

- Gli statement di tipo SELECT restituiscono un result set, mentre gli statement di tipo INSERT, DELETE, UPDATE restituiscono il numero delle tuple aggiornate
- Per verificare se lo statement ha restituito un result set:

```
SQLHSTMT hstmt1;
SQLSMALLINT * number;
SQLNumResultCols(hstmt1,&number);
```

  se restituisce 0, allora il result set è vuoto
- Questa informazione può essere utile nel caso in cui lo statement sia costruito dinamicamente

43

Elaborazione risultato

- I campi delle tuple restituite devono essere associati a determinate variabili, nelle quali verranno inseriti i valori durante la scansione

```
SQLHSTMT hstmt1;
SQLSMALLINT  ColumnNumber;
SQLSMALLINT  TargetType;
SQLPOINTER   TargetValuePtr;
SQLINTEGER BufferLenght;
SQLINTEGER* Info;
SQLBindCol(hstmt1,ColumnNumber,TargetType,
                TargetValuePtr,BufferLenght,Info);
```

44

Elaborazione risultato

- `ColumnNumber`: numero progressivo colonna della tabella
- `TargetType`: tipo C nel quale convertire il tipo SQL
- `TargetValuePtr`: puntatore al campo nel quale inserire il valore estratto dalla tupla
- `BufferLenght`: lunghezza in byte `TargetValuePtr`
- `Info`: buffer (di tipo intero) nel quale l'istruzione `SQLFetch` inserirà informazioni circa il risultato dell'operazione di binding:
  - la lunghezza in byte del campo da inserire nella variabile
  - `SQL_NULL_DATA`: il campo contiene NULL
  - `SQL_NO_DATA`: nessuna tupla letta

45

Esempio di binding

```
SQLCHAR Matr[4];
SQLCHAR Nome[20];
SQLCHAR Cognome[20];
SQLCHAR Manager[4];
SQLINTEGER Stipendio, Matr_info,Nome_info,Cognome_info,
                    Indirizzo_info,Manager_info,Stipendio_info;
SQLRETURN re;
SQLHSTMT hstmt;

SQLBindCol(hstmt, 1, SQL_C_CHAR, Matr, sizeof(Matr), Matr_info);
SQLBindCol(hstmt, 2, SQL_C_CHAR, Nome, sizeof(Nome), Nome_info);
SQLBindCol(hstmt, 3, SQL_C_CHAR, Cognome, sizeof(Cognome),
   Cognome_info);
SQLBindCol(hstmt, 4, SQL_C_CHAR, Manager, sizeof(Manager),
   Manager_info);
SQLBindCol(hstmt, 5, SQL_C_CHAR, Stipendio, sizeof(Stipendio),
   Stipendio_info);
```

46

Elaborazione risultato

- Un cursore viene automaticamente creato quando viene eseguito uno statement che crea un result set
- Simile all'SQL da programma
- Un cursore è una sorta di puntatore che si sposta sulle tuple contenute nel result set
- L'istruzione che ci permette di muoverci sulla tupla successiva è

```
SQLHSTMT hstmt1;
SQLFetch(hstmt1);
```

- Questa istruzione può restituire:
  - `SQL_SUCCESS`: la tupla è stata correttamente letta
  - `SQL_ERROR`: si è verificato un errore
  - `SQL_NO_DATA`: non ci sono più tuple

47

Elaborazione risultato

- Se lo statement non ha creato alcun result set, `SQLFetch` provoca un errore
- Al termine della scansione il cursore deve essere esplicitamente chiuso

```
SQLHSTMT hstmt1;
SQLCloseCursor(hstmt1);
```

48

Esempio

```
while((re= SQLFetch(hstmt)) != SQL_NO_DATA)
{
  if(re ==SQL_ERROR..) {".segnala errore ".. };
  if(re ==SQL_SUCCESS..)
  {
      printf(" Nome: %s %s, Manager: %s, Stipendio: %d ",
                 Nome,Cognome,Manager, Stipendio);
  }
}

SQLCloseCursor(hstmt);
```

49

Elaborazione risultato

- Se lo statement è di tipo INSERT, DELETE, UPDATE si può avere la necessità di determinare il numero di tuple aggiornate
- nel momento in cui lo statement viene eseguito, il numero di tuple aggiornate viene mantenuto in una variabile di sistema
- per recuperarlo:

```
SQLHSTMT hstmt1;
SQLSMALLINT  number;
SQLRowCount(hstmt1,&number);
```

50

Passo 5: Commit transazione

- È possibile richiedere:
  - il commit o l'abort di tutti gli statement associati ad una certa connessione
    ```
    SQLHDBC hdbc1;
    SQLEndTrans(SQL_HANDLE_DBC,hdbc1,SQL_COMMIT);
    SQLEndTrans(SQL_HANDLE_DBC,hdbc1,SQL_ABORT);
    ```
  - di tutte le connessioni associate ad un ambiente
    ```
    SQLHENV henv1;
    SQLEndTrans(SQL_HANDLE_ENV,henv1,SQL_COMMIT);
    SQLEndTrans(SQL_HANDLE_ENV,henv1,SQL_ABORT);
    ```

51

Passo 6: Disconnessione

- Viene liberato lo statement
  ```
  SQLHSTMT hstmt1;
  SQLFreeHandle(SQL_HANDLE_STMT,hstmt1);
  ```
- disconnessione
  ```
  SQLHDBC hdbc1;
  SQLDisconnect(hdbc1);
  ```
- viene liberato il buffer di connessione
  ```
  SQLHDBC hdbc1;
  SQLFreeHandle(SQL_HANDLE_DBS,hdbc1);
  ```
- viene liberato l'ambiente
  ```
  SQLHENV henv1;
  SQLFreeHandle(SQL_HANDLE_ENV,henv1);
  ```

52

Esempio complessivo

Esempio ODBC con sintassi C:

```
SQLHENV     henv;       /* variabile ambiente
SQLHDBC     hdbc;       /* variabile buffer transito dati
SQLHSTMT hstmt;         /* variabile comando SQL
SQLRETURN   re;            /* valore di ritorno

SQLCHAR Matr[4];
SQLCHAR Nome[20];
SQLCHAR Cognome[20];
SQLCHAR Manager[4];
SQLINTEGER Stipendio, Matr_info,Nome_info,Cognome_info,
           Indirizzo_info,Manager_info,Stipendio_info;
SQLRETURN re;
SQLHSTMT hstmt;
```

53

Esempio complessivo (continua)

```
int main ()
{
re = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,  &henv);

if(re == SQL_SUCCESS) {
  re = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);

   if(re == SQL_SUCCESS) {
      re = SQLConnect(hdbc, ”ContiCorrenti”, SQL_NTS,
            ”Pippo”, SQL_NTS, ”password”, SQL_NTS);
      if (re == SQL_SUCCESS) {
        re = SQLExectDirect(hstm, ”SELECT* FROM Impiegati”);
             if (re == SQL_SUCCESS){
```

54

Esempio complessivo (continua)

```
SQLBindCol(hstmt, 1, SQL_C_CHAR, Matr, sizeof(Matr), Matr_info);
SQLBindCol(hstmt, 2, SQL_C_CHAR, Nome, sizeof(Nome), Nome_info);
SQLBindCol(hstmt, 3, SQL_C_CHAR, Cognome, sizeof(Cognome),
   Cognome_info);
SQLBindCol(hstmt, 4, SQL_C_CHAR, Manager, sizeof(Manager),
   Manager_info);
SQLBindCol(hstmt, 5, SQL_C_CHAR, Stipendio, sizeof(Stipendio),
   Stipendio_info);

while((re= SQLFetch(hstmt)) != SQL_NO_DATA){
   if(re ==SQL_ERROR..) {" .. segnala errore ".. };
   if(re ==SQL_SUCCESS..)
         { printf("Nome: %s %s, Manager: %s, Stipendio: %d",
               Nome,Cognome,Manager, Stipendio);}
    }
SQLCloseCursor(hstmt);
SQLEndTrans(SQL_HANDLE_DBC,hdbc,SQL_COMMIT);
```

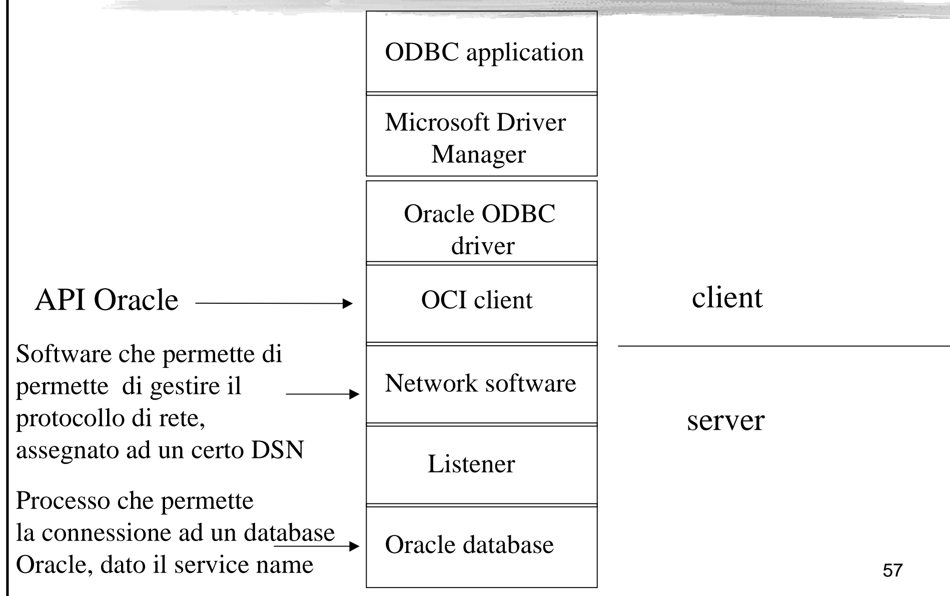
55

Esempio complessivo (continua)

```
SQLFreeHandle(SQL_HANDLE_STMT,hstmt);
};
SQLDisconnect(hdbc);
};
SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
};
SQLFreeHandle(SQL_HANDLE_ENV,henv);
}
}
```

56

ODBC in Oracle



57

ODBC in Oracle

- Livelli di conformità:
  - API: Core + insieme limitato funzionalità livelli 1 e 2
  - SQL: di base (Entry level) + alcune funzionalità Intermediate e Full level
- Due concetti:
  - service name: identifica il database (server + protocollo di comunicazione)
  - DSN: tipo sorgente dati (Oracle) + service name
- Passi:
  - si installa Oracle ODBC driver sul client
  - si impostano i service name
  - si imposta DSN
  - si può usare il driver con la DSN definita

58

JDBC (Java Database Connectivity)

- ODBC è un'API sviluppata in C che richiede API intermedie per essere utilizzata con altri linguaggi
- Inoltre è difficile da capire: alcuni concetti vengono portati a livello interfaccia ma sarebbe meglio mantenerli nascosti
- JDBC (che non è solo un acronimo ma un trademark della SUN) è stato sviluppato nel 1996 dalla Sun per superare questi problemi
- è compatibile ed estende X/open SQl CLI e ISO SQL/CLI

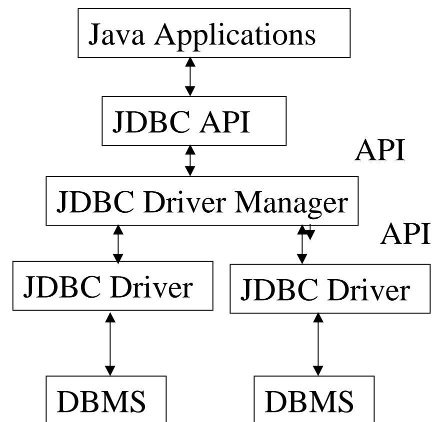
59

JDBC

- Rappresenta una API 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

60

Architettura generale



61

Dove si trova?

- JDBC 1 inclusa in JDK 1.1 (Java Development Kit)
- JDBC 2 inclusa in JDK 1.2
  - include tipi di dati SQL 3 (object-relational)
  - operazioni batch
  - ...
- Attualmente: JDBC 3 in via di approvazione
- JDK è gratuito
- Installando JDK si installa Driver Manager
- Driver: nessuna installazione particolare (copia)

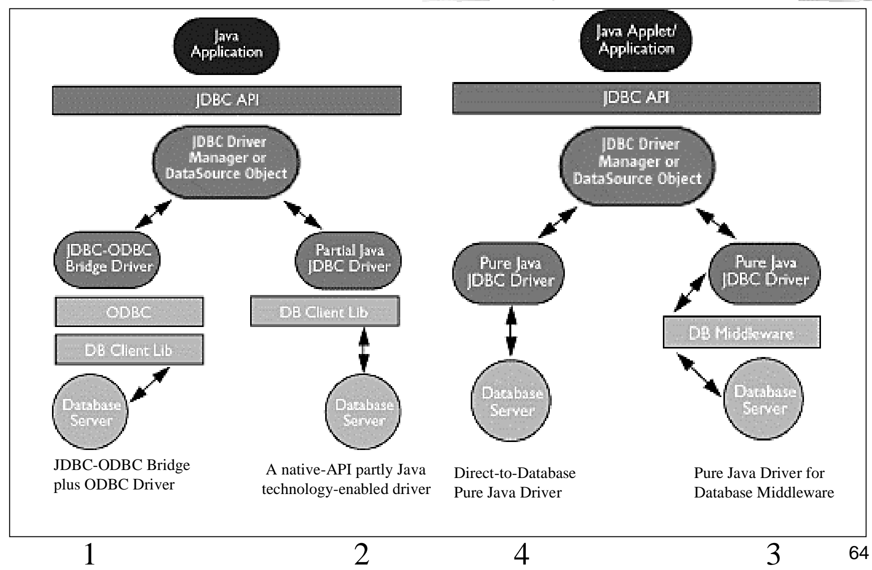
62

Tipi di driver

- Esistono quattro tipi di driver:
  - JDBC-ODBC Bridge + ODBC Driver
  - A native-API partly Java technology-enabled driver
  - Pure Java Driver for Database Middleware
  - Direct-to-Database Pure Java Driver.

63

Tipi di driver



64

Driver di tipo 1

- Accesso a JDBC tramite un driver ODBC
- Uso di JDBC-ODBC bridge
- il codice ODBC binario e il codice del DBMS client, deve essere caricato su ogni macchina client che usa JDBC-ODBC bridge
- si consiglia di utilizzare questa strategia quando non esistono soluzioni alternative …
- è necessario installare le librerie sul client perché non sono trasportabili su HTTP
- compromette “write once, run anywhere”

65

Driver di tipo 2

- Le chiamate a JDBC vengono tradotte in chiamate all’API del DBMS prescelto (es. OCI Oracle)
- Il driver contiene codice Java che chiama metodi scritti in C/C++
- non utilizzabile per applet/servlet
- anche in questo caso, codice binario deve essere caricato sul client
- compromette “write once, run anywhere”

66

Driver di tipo 3

- Basato completamente su Java (utilizzabile con Applet/servlet)
- converte le chiamate JDBC nel protocollo di una applicazione middleware che traduce quindi la richiesta del client nel protocollo proprietario del DBMS
- l'applicazione middleware può garantire l'accesso a diversi DBMS
- il protocollo middleware dipende dal DBSM sottostante

67

Driver di tipo 4

- Basato completamente su Java (utilizzabile con Applet/servlet)
- converte le chiamate JDBC nel protocollo di rete usato direttamente dal DBMS
- permette quindi un accesso diretto dalla macchina client alla macchina server
- è la soluzione più pura in termini Java

68

Caratteristiche e benefici

Benefici:
- Flessibilità
- semplice ed economico
- non è richiesta alcuna configurazione a livello client

Caratteristiche:
- installazione semplice
- facile accesso ai metadati (non li vediamo)
- possibilità di accedere database tramite URL

69

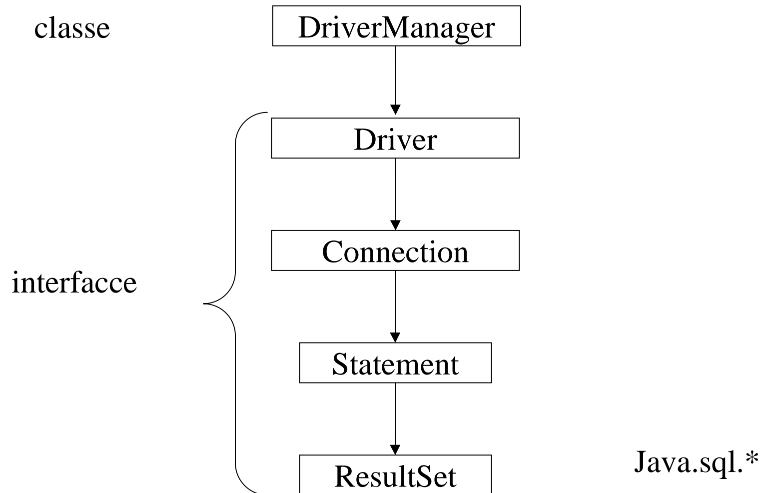
Tipi di dato

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

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

70

Interfaccia JDBC (semplificata - solo le classi che vedremo)



71

Passo 1: Caricamento driver

- Il driver manager mantiene una lista di classi che implementano l'interfaccia java.sql.Driver
- l'implementazione del Driver deve essere registrata in qualche modo nel DriverManager
- JDBC richiede che, quando una classe Driver viene caricata, crei un'instanza ed effettui anche la registrazione

72

Passo 1: caricamento driver

- Due modi:
  - `Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newinstance();`
  - `newinstance()` si può omettere se viene direttamente effettuata l'inizializzazione
  - `DriverManager.registerDriver(new`
    `sun.jdbc.odbc.JdbcOdbcDriver());`
- Il nome della classe da usare viene fornito con la documentazione relativa al driver

73

Esempio di caricamento driver

```
import java.sql.*;

class JdbcTest
{
   public static void main (String args []) {
      Class.forName ("oracle.jdbc.OracleDriver");
   }
}
```

74

Passo 2: Connessione

- Per realizzare la connessione vengono utilizzate le seguenti classi ed interfaccie:
  - classe `java.sql.DriverManager`: gestisce la registrazione dei driver
  - interfaccia `java.sql.Driver`: non viene esplicitamente utilizzata a livello applicativo
  - interfaccia `java.sql.Connection`: permette di inviare una serie di richieste SQL al DBMS
- È possibile connettersi a qualunque database, locale e remoto, specificandone l'URL
- in JDBC, l'URL è formato da tre parti:

  jdbc: <subprotocol>: <subname>
  - <subprotocol> identifica il driver o il meccanismo di connessione al database
  - <subname> dipende da subprotocol ed identifica lo specifico database

Connessione

- Esempi:
  - jdbc:oracle:thin:@everest:1521:GEN:
    - subprotocol: Oracle
    - subname:
      - thin specifica che deve essere utilizzari Oracle ODBC Thin driver(vedere oltre)
      - Everest specifica il nome della macchina
      - 1521: numero porta
      - GEN: nome database Oracle
  - `jdbc:mysql://cannings.org:3306/test`
    - subprotocol: MySQL
    - subname:
      - `cannings.org` specifica il nome della macchina
      - `3306` : numero porta
      - `test` : nome database MySQL
  - se si usa JDBC-ODBC driver: `jdbc:odbc:subname`

Connessione

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

```
Connection con =
  DriverManager.getConnection("jdbc:mysql://cannings.org:3
  306/test", "myLogin", "myPassword");
```

- Se uno dei driver caricati riconosce l'URL fornito dal metodo, il driver stabilisce la connessione

77

Esempio di connessione

```
import java.sql.*;

class JdbcTest
{
   static String ARS_URL =
   "jdbc:oracle:@PutDatabaseNameHere";

   public static void main (String args [])
   {
      Class.forName ("oracle.jdbc.OracleDriver");
      Connection ARS;
      ARS =DriverManager.getConnection(ARS_URL,
                        "whitney",
                        "secret");
   }
}
```

78

Passo 3: creazione ed esecuzione statement

- Per creare ed eseguire uno statement vengono utilizzate le seguenti classi:
  - `java.sql.Connection`: per creare lo statement
  - `java.sql.Statement`: permette di eseguire gli statement
  - `java.sql.ResultSet`: per analizzare i risultati delle query
- un oggetto di tipo Statement viene creato a partire da un oggetto di tipo Connection e permette di inviare comandi SQL al DBMS :

```
Connection con;
…
Statement stmt = con.createStatement();
```

79

Esecuzione statement

- Come in ODBC, è nenessario distinguere tra statement che rappresentano query e statement di aggiornamento
- Per eseguire una query:

```
stmt.executeQuery(”SELECT * FROM IMPIEGATI");
```

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

```
stmt.executeUpdate(”INSERT INTO IMPIEGATI VALUES
‘AB34’,‘Gianni’, ‘Rossi’,’GT67’,1500");

stmt.executeUpdate("CREATE TABLE PROVA (CAMPO1     NUMBER)”);
```

- il terminatore dello statement (es. ‘;’) viene inserito direttamente dal driver prima di sottomettere lo statement al DBMS per l’esecuzione

80

Prepared Statement

- Come in ODBC, è possibile preparare gli statement prima dell'esecuzione
- Questo garantisce
  - statement precompilato
  - riduce i tempi di esecuzione
  - usate per statement utilizzati molte volte

```
PreparedStatement queryImp = con.prepareStatement(
                        "SELECT * FROM IMPIEGATI");

queryImp.executeQuery():
```

81

Uso di parametri

- È possibile specificare che la stringa che rappresenta lo statement SQL deve essere completata con parametri (eventualmente letti da input) identificati da '?'

```
PreparedStatement queryImp = con.prepareStatement(
             "SELECT * FROM IMPIEGATI WHERE Nome = ?");

queryImp.setString(1, 'Rossi');
queryImp.executeQuery():
```

- Questo è possibile anche in ODBC (non visto)
- Si noti l'uso di " e ' (devono essere alternati)
- setXXX, dove XXX è il tipo Java del valore del parametro

82

Passo 4: Elaborazione risultato

- JDBC restituisce i risultati di esecuzione di una query in un result set, come ODBC

```
String query = " SELECT * FROM IMPIEGATI ";
ResultSet rs = stmt.executeQuery(query);
```

- Come in ODBC, il result set è costruito solo per query e non per INSERT, DELETE, UPDATE
- In questo caso viene restituito un intero, che rappresenta il numero di tuple modificate
- Per muoversi su un result set si deve utilizzare il metodo next:

```
while (rs.next()) {
    String s = rs.getString("Cognome");
    float n = rs.getFloat("Stipendio");
    System.out.println(s + "   " + n);
}
```

83

Metodi per accedere i valori associati agli attributi

- Il metodo `getXXX`, di un `ResultSet`, permette 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 posizionali:

```
String s = rs.getString(2);
int n = rs.getInt(5);
```

- Usare `getInt` per valori numerici, `getString` per char, varchar

84

Metodi per accedere i valori associati agli attributi

- Il metodo next() permette di spostarsi nel result set (cursore):

```
while (rs.next()) {/* get current row */}
```

- inizialmente il cursore è posizionato prima della prima tupla
- il metodo diventa falso quando non ci sono più tuple

85

Esempio esecuzione diretta

```
import java.sql.*;
import java.io.*;
class JdbcTest
{
   static String ARS_URL = "jdbc:oracle:@PutDatabaseNameHere";
   public static void main (String args [])
{
      Class.forName ("oracle.jdbc.OracleDriver");
      Connection ARS;
      ARS =DriverManager.getConnection(ARS_URL,
                        "whitney", "secret");

      Statement selImp = ARS.createStatement ();

      String stmt =  "SELECT * FROM Impiegati WHERE Cognome ='Rossi'";

      ResultSet  impRossi = selImp.executeQuery (stmt);
      while ( impRossi.next() )
      {
         System.out.println (impRossi.getString ("Stipendio"));
      }
   } }
```

86

Esempio prepared statement

```
import java.sql.*;
import java.io.*;
class JdbcTest
{
   static String ARS_URL = "jdbc:oracle:@PutDatabaseNameHere";
   public static void main (String args []) {
      Class.forName ("oracle.jdbc.OracleDriver");
      Connection ARS;
      ARS =DriverManager.getConnection(ARS_URL,
                        "whitney", "secret");

      String stmt =
               "SELECT * FROM Impiegati WHERE Cognome = 'Rossi'";

      PreparedStatement prepStmt = ARS.preparedStatement (stmt);
      ResultSet impRossi = prepStmt.executeQuery ();
      while ( impRossi.next() )
     {
        System.out.println (impRossi.getString ("Stipendio"));
     }

}
}
```

87

Passo 5: Transazioni

- Per default, JDBC esegue il commit di ogni statement SQL inviato al DBMS (auto-commit)
- è possibile disattivare l'autocommit tramite il metodo `setAutoCommit`

```
Connection con;
…
con.setAutoCommit(false);
```

- a questo punto ogni connessione diventa una transazione indipendente per la quale si può richiedere il commit o l'abort tramite:

```
con.commit();
con.abort();
```

88

Esempio transazioni

```
import java.sql.*;
import java.io.*;
class JdbcTest
{
   static String ARS_URL = "jdbc:oracle:@PutDatabaseNameHere";
   public static void main (String args [])
{
      Class.forName ("oracle.jdbc.OracleDriver");
      Connection ARS;
      ARS =DriverManager.getConnection(ARS_URL,
                        "whitney", "secret");
      ARS.setAutoCommit(false);
      Statement selImp = ARS.createStatement ();
      String stmt1 =
        "UPDATE Impiegati SET Stipendio = 1000 WHERE Cognome = 'Rossi'";
      String stmt2 =
       "DELETE FROM Impiegati WHERE Cognome = 'Verdi'";

      selImp.executeUpdate (stmt1);
      selImp.executeUpdate (stmt2);

      ARS.commit();
      ARS.close();     } }
```

89

Passo 6: Disconnessione (chiusura)

- Per risparmiare risorse, può essere 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 `ResultSet` associati

90

Eccezioni

- La classe `java.sql.SQLException` estende la classe `java.lang.Exception` in modo da fornire informazioni ulteriori in caso di errore di accesso al database, tra cui:
  - la stringa SQLState che rappresenta la codifica dell'errore in base allo standard X/Open
    - `getSQLState()`
  - il codice di errore specifico al DBMS
    - `getErrorCode()`
  - una descrizione dell'errore
    - `getMessage()`

91

Esempio

```
import java.sql.*;
import java.io.*;
class JdbcTest
{
   static String ARS_URL = "jdbc:oracle:@PutDatabaseNameHere";

   public static void main (String args [])
{   try{
      Class.forName ("oracle.jdbc.OracleDriver");
      Connection ARS;
      ARS =DriverManager.getConnection(ARS_URL,
                        "whitney", "secret");
   }
   catch (SQLException e) {
     while( e!=null){
               System.out.println("SQLState: " + e.getSQLState());
               System.out.println("    Code: " + e.getErrorCode());
               System.out.println(" Message: " + e.getMessage());
               e = e.getNextException();
}}}}
```

92

Warning

- Sottoclasse di `SQLException`
- la classe `java.sql.SQLWarning` fornisce informazioni su warning relativi all'accesso al database
  - i warning vengono collegati agli oggetti i cui metodi hanno generato l'eccezione
    - connection
    - statement
    - resultset
  - recuperabii con il metodo `getWarnings()`
- non bloccano l'esecuzione del programma

93

Esempio

```
Statement selImp = ARS.createStatement ();
String stmt =  "SELECT * FROM Impiegati WHERE Cognome ='Rossi'";
ResultSet  impRossi = selImp.executeQuery (stmt);
while ( impRossi.next() )
      {
         System.out.println (impRossi.getString ("Stipendio"));
         SQLWarning warning_stmt = selImp.getWarnings();
         while (warning_stmt != null)
                  {
                       System.out.println("Message: " +
                                      warning_stmt.getMessage());
                       System.out.println("SQLState: " +
                                      warning_stmt.getSQLState());
                       System.out.println("Vendor error code: " +
                                       warning_stmt.getErrorCode());
                       warning_stmt = warning_stmt.getNextWarning();

                 }
       }
```

94

Esempio

```
SQLWarning warning_rs = rs.getWarnings();
while (warning_rs != null)
        {
            System.out.println("Message: " +
                          warning_rs.getMessage());
            System.out.println("SQLState: " +
                          warning_rs.getSQLState());
            System.out.println("Vendor error code: " +
                           warning_rs.getErrorCode());
            warning_rs = warning_rs.getNextWarning();

        }
```

95

JDBC in Oracle

- Quattro tipi di JDBC drivers:
  - uguali dal punto di vista dell'API supportata
  - diversi dal punto di vista della connessione al database
- JDBC Thin driver:
  - 100% pure Java
  - driver di tipo IV
  - usato in applet
  - non richiede software Oracle addizionale sul client
- JDBC OCI driver:
  - driver di tipo II
  - Oracle client deve essere installato sul client
  - non adatti ad applet perché dipendono dalla piattaforma Oracle
  - combinazione di Java e C
  - convertono chiamate JDBC in chiamate OCI

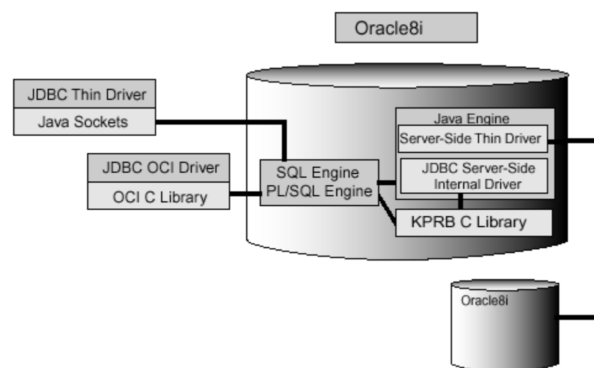
96

JDBC in Oracle

- JDBC Server-side Thin Driver:
  - stesse funzionalità di JDBC Thin driver ma viene eseguito in un database Oracle e può accedere un database remoto
  - utile per accedere un server Oracle remoto da un server Oracle
- JDBC Server-side Internal Driver:
  - comunicazione diretta con il motore SQL del server sul quale è installato
  - no accesso remoto

97

JDBC in Oracle



98