



JDBC

L'interfaccia ResultSet



L'interfaccia ResultSet

- Un ResultSet è un oggetto JDBC che contiene il risultato di una query di selezione
 - Sono le righe e le colonne che soddisfano la condizione dell'istruzione SQL
- Gli oggetti ResultSet sono il punto di connessione tra le tabelle di un database e le applicazioni Java/JDBC
 - Possono rappresentare
 - Una singola tabella (o parte di essa)
 - Più tabelle (risultato di una JOIN)

L'interfaccia ResultSet

- ResultSet dispone di metodi per accedere alle righe e alle colonne rappresentate nell'oggetto
- In base all'implementazione del driver e alle esigenze delle applicazioni un ResultSet può essere:
 - Dal punto di vista dell'accesso:
 - “forward only” : accesso sequenziale alle righe (l'unico consentito con JDBC 1.0)
 - “scrollabile”: accesso casuale (introdotto con JDBC 2.0)
 - Dal punto di vista della modifica:
 - “solo lettura”
 - “modificabile”



L'interfaccia ResultSet

- L'interfaccia ResultSet incapsula l'operatore *cursore*
- Il cursore si occupa di gestire le righe provenienti da una query di selezione
- Con i cursori è possibile mantenere allineati i dati tra DBMS e ResultSet
- I driver possono
 - usare una propria implementazione dei cursori a livello client
 - sfruttare quella presente in molti DBMS (cursori lato server)

Creare un ResultSet

- Gli oggetti *ResultSet* sono costruiti a partire da oggetti *Statement*, *PreparedStatement* e *CallableStatement* che lavorano su query di selezione
- Per default gli oggetti *ResultSet* sono creati di tipo *forward only* e *non modificabili*
 - Una volta visualizzate le righe di un *ResultSet* non è più possibile accedervi a meno di non rieseguire la query e creare un nuovo *ResultSet*
- Esempio: dopo aver creato un oggetto *Statement* è possibile creare un *ResultSet* con il metodo *executeQuery*

Esempio con Statement

```
Statement st = con.createStatement();
```

```
ResultSet rs =  
    st.executeQuery("SELECT * FROM articolo " +  
        "WHERE prezzo<50");
```

```
while(rs.next()) {  
    System.out.println("Codice = " +  
        rs.getString("codice"));  
    System.out.println("Descrizione = " +  
        rs.getString("descrizione"));  
    System.out.println("Prezzo = " +  
        rs.getString("prezzo"));  
}
```

Esempio con PreparedStatement

```
PreparedStatement pst =  
    con.prepareStatement("SELECT * FROM articolo " +  
        "WHERE prezzo<50");
```

```
ResultSet rs = pst.executeQuery();
```

```
while(rs.next()) {  
    System.out.println("Codice = " +  
        rs.getString("codice"));  
    System.out.println("Descrizione = " +  
        rs.getString("descrizione"));  
    System.out.println("Prezzo = " +  
        rs.getString("prezzo"));  
}
```

/ si noti che viene usato executeQuery senza parametri perchè con PreparedStatement la query è già impostata in fase di costruzione dell'oggetto*/*



Navigazione nel RecordSet

Muoversi tra le righe

- Un cursore può essere visto come un puntatore alla riga corrente nel *ResultSet*
 - Quando il *ResultSet* è creato con *executeQuery*, il cursore sarà prima della prima riga
 - La chiamata al metodo *next()* provoca il posizionamento sulla prima riga (e quindi la possibilità di accedere ai dati)
 - *next()* risponde true finchè il cursore punta ad una riga valida.
 - *next()* restituisce falso non appena il cursore si posiziona dopo l'ultima riga
- Ogni tentativo di accedere agli attributi quando il cursore si trova prima della prima riga o dopo l'ultima riga provocherà una *SQLException*

RecordSet navigabili

- Con la versione 2.0 di JDBC sono stati introdotti metodi (diversi da *next*) che consentono il movimento tra le righe in tutte le direzioni e in modo casuale
- Se vogliamo *ResultSet* “scrollabili” è necessario che l’oggetto *Statement* di partenza ne sia informato
- Occorre utilizzare un’altra versione di *createStatement* che ha come argomento:
 - il tipo di *ResultSet* e
 - il tipo di concorrenza

```
Statement st = con.createStatement  
    (ResultSet.TYPE_SCROLL_SENSITIVE,  
     ResultSet.CONCUR_UPDATABLE) ;
```

```
ResultSet rs = st.executeQuery("SELECT * FROM articolo");
```

Metodi per la navigazione

- *boolean next()*

- ☐ muove il puntatore alla riga successiva
- ☐ restituisce false se viene raggiunta la fine del *ResultSet* e il cursore punterà dopo l'ultima riga

- *boolean previous()*

- ☐ muove il puntatore alla riga precedente
- ☐ restituisce false se viene raggiunto l'inizio del *ResultSet* e il cursore punterà prima della prima riga



Metodi per la navigazione

- *boolean first()*

- ☐ muove il puntatore alla prima riga
- ☐ restituisce false se il *ResultSet* non ha nessuna riga

- *boolean last()*

- ☐ muove il puntatore all'ultima riga
- ☐ restituisce false se il *ResultSet* non ha nessuna riga

Metodi per la navigazione

- *boolean absolute(int n)* con $n \neq 0$
 - Se $n > 0$ il puntatore punta all' n -esima riga del *ResultSet* (a partire dalla prima)
 - Se $n < 0$ il puntatore si sposterà n righe indietro a partire dalla fine del *ResultSet*
 - Se si raggiunge l'inizio o la fine del *ResultSet* restituisce false
- *boolean relative(int n)*
 - Se $n > 0$ il puntatore si sposterà di n righe avanti nel *ResultSet* a partire dall'attuale
 - Se $n < 0$ il puntatore si sposterà n righe indietro a partire dall'attuale
 - $n = 0$ è consentito ma non ha effetto
 - Se si raggiunge l'inizio o la fine del *ResultSet* il metodo restituisce false



Metodi per la navigazione

- *void afterfirst()*

- muove il puntatore immediatamente prima della prima riga

- *void afterlast()*

- muove il puntatore immediatamente dopo dell'ultima riga

Navigazione: Esempio

```
Statement st =  
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
    ResultSet.CONCUR_READ_ONLY);
```

```
ResultSet rs =  
    st.executeQuery("SELECT * FROM articolo WHERE prezzo < 50");
```

```
rs.afterLast();  
while(rs.previous()) {  
    System.out.println("Codice = " + rs.getString("codice"));  
    System.out.println("Descrizione = " + rs.getString("descrizione"));  
    System.out.println("Prezzo = " + rs.getFloat("prezzo"));  
}
```

```
rs.absolute(5);  
while(rs.next()) {  
    System.out.println("Codice = " + rs.getString("codice"));  
    System.out.println("Descrizione = " + rs.getString("descrizione"));  
    System.out.println("Prezzo = " + rs.getString("prezzo"));  
}
```

Metodi per controllare il cursore

- *boolean isFirst()*
 - Restituisce true se il puntatore è sulla prima riga del *ResultSet*
- *boolean isBeforeFirst()*
 - Restituisce true se il puntatore è immediatamente prima della prima riga del *ResultSet*
- *boolean isLast()*
 - Restituisce true se il puntatore è sull'ultima riga del *ResultSet*
- *boolean isAfterLast()*
 - Restituisce true se il puntatore è immediatamente dopo l'ultima riga del *ResultSet*
- *int getRow()*
 - Restituisce il numero della riga corrente



Accesso ai campi

Metodi *getXXX*

- I metodi *getXXX* operano sulla riga puntata dal cursore e consentono di leggere i valori associati alle colonne del *ResultSet* (cioè agli attributi)
 - *XXX* rappresenta il tipo Java da usare per gestire il valore
 - *getString*, *getInt*, *getFloat*, ecc...

Metodi getXXX

- E' possibile accedere ad una colonna specificandone il nome come parametro di *getXXX*

- Esempio:

```
String codice = rs.getString("codice");
```

dove *codice* è una stringa non case-sensitive

Metodi getXXX

- E' possibile accedere ad una colonna anche specificando la posizione della colonna all'interno del *ResultSet*
- Esempio:
Se la query è:

```
SELECT codice, genere, prezzo,  
FROM articolo
```


allora

```
rs.getString(1); ↔ rs.getString("codice");
```
- Se la query utilizza * (non esplicita i nomi delle colonne), allora la numerazione segue l'ordine presente nella tabella
- Se la query è una JOIN allora conviene usare gli indici invece dei nomi perché ci potrebbero essere colonne con lo stesso nome

Metodi getXXX

- Sappiamo che i tipi SQL sono mappati in tipi JDBC/SQL ognuno dei quali corrisponde a uno o più tipi Java
- I metodi getXXX restituiscono un tipo o classe Java, mentre i dati delle tabelle hanno una rappresentazione legata al tipo JDBC/SQL
- Esempio: Data l'istruzione:

```
String codice = rs.getString("codice");
```


l'applicazione cercherà di convertire il valore dell'attributo codice in una String Java

Metodi getXXX

- Esempio: l'istruzione

```
int short = rs.getShort("codice");
```

può fallire se *codice* nel database:

- ☐ non è di tipo numerico (es: CHAR, o VARCHAR) e il valore non può essere convertito in un numero (contiene caratteri alfanumerici),
 - ☐ se di tipo numerico ma con un valore superiore al range consentito per gli short
- E' responsabilità del programmatore assicurarsi che le conversioni non falliscano e che non ci sia perdita di precisione



Tipi di ResultSet

Tipi di ResultSet

A. *ResultSet.TYPE_FORWARD_ONLY*

- Non scrollabile: il cursore si può muovere solo in avanti con il metodo *next* dalla prima all'ultima
- Vantaggio:
 - Assicura max portabilità sia tra le diverse versioni sia di JDBC, sia tra i DBMS
- Svantaggio:
 - A. Limitazione delle funzionalità

B. *ResultSet.TYPE_SCROLL_INSENSITIVE*

- Scrollabile (con i metodi *previous*, *first*, *last*, ecc)
- Non si accorge di eventuali modifiche che avvengono alla tabella associata per opera di altri processi

C. *ResultSet.TYPE_SCROLL_SENSITIVE*

- Scollabile (con i metodi *previous*, *first*, *last*, ecc)
- Si accorge di eventuali modifiche alla tabella associata (qualsiasi modifica alle tabelle dopo la creazione del *ResultSet* sarà visibile)

■ Warning:

- Non tutti i driver supportano ResultSet di Tipo B e C
- `boolean supportResultSetType(int type)`
dell'interfaccia *DatabaseMetaData* restituisce vero se il ResultSet del tipo specificato è supportato dal driver corrente

Tipi di Concorrenza

A. *ResultSet.CONCUR_READ_ONLY*

- E' di sola lettura non può fare modifiche sul database
- Con JDBC 1.0 era il solo tipo di concorrenza consentita
- Vantaggio:
 - Concorrenza illimitata: un lock di sola lettura non impedisce che n client possano accedere contemporaneamente alla stessa riga della tabella
- Svantaggio:
 - A. Limitazione delle funzionalità

B. *ResultSet.CONCUR_UPDATABLE*

- I ResultSet possono essere utilizzati per aggiornare i dati presenti nelle tabelle del DBMS
- Introdotto con JDBC 2.0
- Vantaggio:
 - È possibile direttamente da programma inserire, cancellare aggiornare righe nelle tabelle senza ricorrere a INSERT INTO, DELETE FROM o UPDATE
- Svantaggio:
 - Riduce il livello di concorrenza:
 - Con un lock write-only una riga potrebbe essere inaccessibile ad altri processi

ResultSet: Esempio

- Con oggetti Statement

```
Statement st = con.createStatement  
    (ResultSet.TYPE_SCROLL_INSENSITIVE,  
     ResultSet.CONCUR_READ_ONLY);  
ResultSet rs = st.executeQuery("SELECT * FROM  
articolo ");
```

- Con oggetti PreparedStatement

```
PreparedStatement prepareStatement  
    (String sql, int ResultSetType, int ResultSetConcurrency);
```

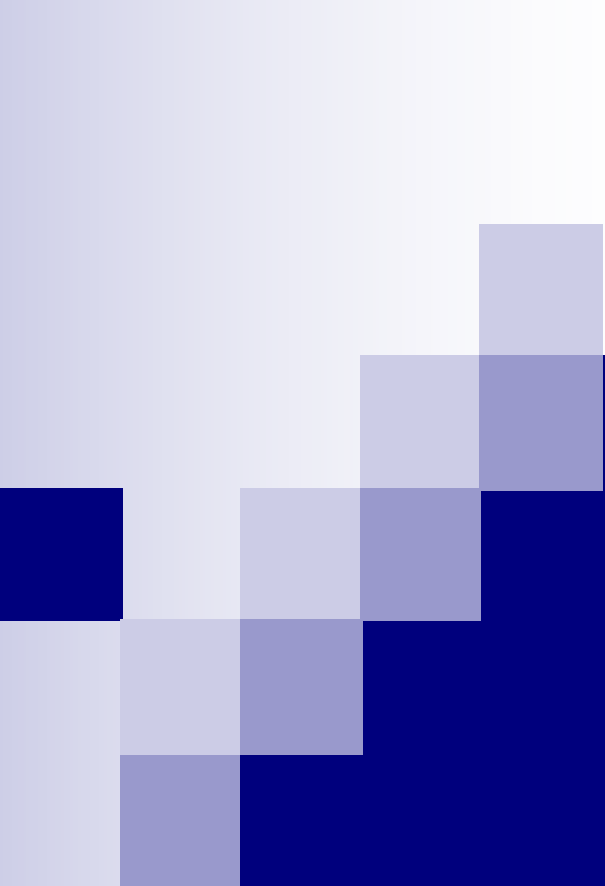
- Con oggetti CallableStatement

```
CallableStatement prepareCall  
    (String sql, int ResultSetType, int ResultSetConcurrency);
```

ResultSet: Esempio

- Con oggetti PreparedStatement

```
PreparedStatement pst =  
con.prepareStatement  
    ("SELECT * FROM articolo WHERE prezzo < ?",  
     ResultSet.TYPE_SCROLL_SENSITIVE,  
     ResultSet.CONCUR_UPDATABLE);  
  
pst.setInt(1, 50);  
ResultSet rs = pst.executeQuery();
```



ResultSet Aggiornabili

Metodi *updateXXX*

- Operano sulla riga puntata dal cursore e consentono di inserire un nuovo valore in un attributo
 - Hanno due parametri:
 - il primo specifica l'attributo (il nome della colonna o l'indice)
 - il secondo il nuovo valore

```
void updateXXX(int columnIndex, XXX newValue);  
void updateXXX(String columnName, XXX newValue);
```

- XXX rappresenta il tipo o classe Java
 - Valgono le stesse regole di conversione date per *getXXX*

```
void updateInt(int columnIndex, int newValue);  
void updateInt(String columnName, int newValue);
```

Metodi *updateXXX*

- Permettono opzionalmente di salvare le modifiche sul db.
- Il cambiamento effettivo nel database avverrà solo dopo aver invocato il metodo *updateRow* *che opera sulla riga corrente!*

Metodi *updateXXX*: Esempio

```
Statement st = con.createStatement  
    (ResultSet.TYPE_SCROLL_INSENSITIVE,  
     ResultSet.CONCUR_UPDATABLE) ;
```

```
ResultSet rs = st.executeQuery  
    ("SELECT codice, descrizione, prezzo" +  
     "FROM articolo " +  
     "WHERE codice = 'AB 0777'");
```

```
if (rs.next()) {  
    rs.updateFloat("prezzo", (float) 28.79);  
    rs.updateRow();  
}
```

```
/* in alternativa  
rs.updateFloat(3, (float) 28.79);*/
```


Metodo *cancelRowUpdates*: Esempio

```
Statement st = con.createStatement  
    (ResultSet.TYPE_SCROLL_INSENSITIVE,  
     ResultSet.CONCUR_UPDATABLE) ;
```

```
ResultSet rs = st.executeQuery  
    ("SELECT * FROM articolo " +  
     "WHERE codice = 'AB 0777'");
```

```
if (rs.next()) {  
    rs.updateFloat("prezzo", (float) 28.79);  
    rs.updateString("codice", "AB 0989");  
  
    rs.cancelRowUpdates();  
}
```

/* prima di invocare `updateRow` è possibile riportare il `ResultSet` ai valori precedenti all'aggiornamento */

Cancellazione di righe

- Il metodo *deleteRow* dell'interfaccia *ResultSet* elimina la riga puntata dal cursore
- Ha effetto sia sul *ResultSet*, sia sulla tabella associata nel db

```
rs.absolute(5) ;
```

```
*/posiziona il cursore sulla quinta riga del ResultSet*/
```

```
rs.deleteRow() ;
```

Inserimento di righe

- Concetto di “*insert row*”
 - Riga che presenta le stesse colonne del *ResultSet* ma non ne fa parte
 - È usato come buffer per memorizzare valori di una riga
 - Il metodo *insertRow* renderà effettivo l’inserimento della riga sia nel *ResultSet* sia nel database
- Passi da seguire:
 - 1) Invocare il metodo *moveToInsertRow* per spostare il cursore sulla *insert row*
 - 2) Usare il metodo *updateXXX* per inserire un valore per tutte le colonne presenti nel *ResultSet*
 - 3) Invocare il metodo *insertRow* per rendere effettivo l’inserimento
 - 4) Invocare il metodo *moveToCurrentRow* per muovere il cursore su una riga del *ResultSet*

Inserimento righe: Esempio

```
Statement st = con.createStatement  
    (ResultSet.TYPE_SCROLL_INSENSITIVE,  
     ResultSet.CONCUR_UPDATABLE);
```

```
ResultSet rs =  
    st.executeQuery("SELECT * FROM articolo");
```

```
rs.moveToInsertRow();
```

```
rs.updateString("codice", "AB 0989");  
rs.updateString("descrizione", "TV COLOR SONY 20p");  
rs.updateFloat("prezzo", (float) 300);
```

```
rs.insertRow();  
rs.moveToCurrentRow();
```

```
/* sposterà il cursore dalla insert row alla riga puntata prima della chiamata  
   moveToInsertRow*/
```



Inserimento righe: Warning

- È possibile usare i metodi *getXXX* quando il cursore punta alla *insert row* ma solo se a quell'attributo è già stato assegnato un valore tramite *updateXXX*, altrimenti porterà un risultato indefinito
- Prima di invocare *insertRow* è necessario che a tutti gli attributi sia stato assegnato un valore, altrimenti si verifica un'eccezione
- L'invocazione di *insertRow* provoca l'aggiunta della riga sia nel *ResultSet* sia nel database



ResultSet aggiornabili: Considerazioni

- Non tutte le query producono ResultSet aggiornabili anche se specificata la concorrenza `CONCUR_UPDATABLE`
- Dipende dall'implementazione del DBMS
- Comunque conviene attenersi alle seguenti regole sulla query :
 - deve coinvolgere una sola tabella (il risultato di una JOIN in genere non è aggiornabile)
 - non deve contenere la clausola `GROUP BY`
 - deve selezionare gli attributi appartenenti alla chiave primaria



Inserimento Righe

- Affinchè vada a buon fine è necessario che la query:
 - Selezioni tutti gli attributi creati con il vincolo NOT NULL
 - Selezioni tutti gli attributi che non hanno un valore di default

Altri metodi dell'interfaccia ResultSet

- `Statement getStatement();`
 - Restituisce l'oggetto `Statement` che ha prodotto la query
- `int getFetchSize();`
- `void setFetchSize(int rows);`
 - Ogni volta che si muove il cursore non si accede al database: per migliorare le prestazioni saranno lette `n` righe e messe in una cache. Questi metodi restituiscono e impostano il numero di righe che devono essere lette dal database (meglio usare il valore predefinito per il driver)
- `void close();`
 - Consente di rilasciare le risorse occupate dall'oggetto *ResultSet* senza attendere la chiusura automatica (quando l'oggetto *Statement* da cui è stato creato viene chiuso oppure non si fa più riferimento all'oggetto)