SQL – Java

- JDBC
- DAO
- Esempio:
  - https://github.com/egalli64/mjd
    - Maven
    - MySQL 8

JDBC

- Permette di connettere un progetto Java a un database
- Si aggiunge la dipendenza JDBC per il database scelto
  - MySQL: Connector Java per la versione corrente
    - Ad es: mysql-connector-java 8.0.x
  - Oracle: ojdbc8 certificato per JDK 8, ojdbc11 …
- Si scrive codice Java usando le interfacce definite nei package

  java.sql

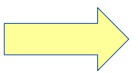
  javax.sql

DriverManager e Connection

- Servizio di base che gestisce i database driver presenti nel progetto
- getConnection()
  - url, segue le specifiche fornite dal DBMS utilizzato
    - jdbc:oracle:thin:@127.0.0.1:1521/xe
    - jdbc:mysql://localhost:3306/me?serverTimezone=Europe/Rome
- Connection
  - Estende l'interfaccia AutoCloseable
  - Media lo scambio di dati tra Java e database

estendendo l'interfaccia AutoCloseable, vuol dire che il mio metodo che invoco sul try una volta finito si chiuderà da solo. Ecco perché su Eclipse l'abbiamo modificato e messo nelle tonde invece che nelle graffe. Se l'avessi lasciato nelle graffe avrei dovuto specificare il "close" in qualche parte nel mio blocco.
Ho fatto un t*ry with resources:*

```
Connection conn = DriverManager.getConnection(url, user, password);
```



Statement

- Rappresenta un comando da eseguire sul database
  - execute() per DDL, true se genera un ResultSet associato
  - executeUpdate() per DML, ritorna il numero di righe interessate
  - executeQuery() per SELECT, ritorna il ResultSet relativo
- Generato da un oggetto Connection per mezzo del metodo createStatement()
- Estende l'interfaccia AutoCloseable
- Lo Statement implica tipicamente quattro passi per il suo svolgimento:
  - Parse, compilazione, pianificazione e ottimizzazione, esecuzione della query

statement:oggetto su cui io posso chiamare metodi, che mi ritornano risultati

1:

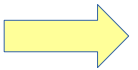
2:

3: rappresentazione Java della tabellina che ritorna quando facciamo una select su mysql. Per eseguire un comando bisogna scriverlo come stringa, in due possibilità, di cui lo statement è quello più semplice.
Il preparedstatement invece ha un jolly che è il punto di domanda, predefinito, in cui noi di volta in volta inseriamo i valori che ci interessano.

PreparedStatement

vantaggio: rendo più rigida la creazione della query che faccio, quindi il chiamante non può mettermi dentro valori sporchi. SQL injection è un attacco al database, e grazie al preparedstatement io rendo più sicuro il mio accesso al database.
In più, posso creare una sola volta il mio statement e modificarlo ogni volta che voglio.

- Estensione di Statement
- Connection.prepareStatement()
  - Uso di '?' come segnaposto per i parametri
- Esecuzione anticipata di parse, compilazione e ottimizzazione della query
- Lo statement può essere precompilato e salvato in cache dal DBMS
- Protezione da attacchi via SQL injection usando i suoi setter per i parametri
- Rende più semplice la conversione di tipi non standard tra Java e SQL

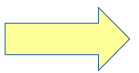


oggetto che rappresenta la tabellina risultante dalla chiamata

ResultSet

- Una tabella di dati che rappresenta il result set ritornato dal database
- Estende l'interfaccia AutoCloseable
- Per default, non supporta update e può essere percorso solo in modalità forward
- Normalmente ottenuto da uno Statement via executeQuery()

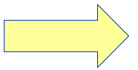
```
ResultSet rs = stmt.executeQuery("SELECT coder_id, first_name, last_name FROM coders");
```



tutti i metodi jdbc possono tirare un'exception. dato che sono molto comuni, tutto il codice viene messo in blocchi try/catch per la sequelexception. il codice che viene fuori è in slide 8

SQLException

- Rappresenta un errore generato da JDBC
- Qualcosa non ha funzionato nell’accesso a database, o altri problemi
- Possiamo assumere che tutto il nostro codice JDBC richieda di essere eseguito in blocchi try/catch per questa eccezione



SELECT via JDBC

try with resources

try sulla risorsa che viene creata. in questo caso connection e statement sono le due risorse, separate dal punto e virgola. La differenza è che lo statement è creato dalla connection, mentre la connection è indipendente (lo si vede da "conn").
il risultato della select viene messo in resultset. Alla riga successiva sto creando una collezione, usando l'arraylist che non ha dimensione predefinita ed è quindi comoda: è un array di stringhe. con next dico al result set di lavorare sulla prossima riga.
ultimo: a result aggiungi il primo elemento della riga corrente del rsultset letta come se fosse una stringa. che voglio la stringa come risultato, glielo devo specificare. l'1 tra parentesi (la colonna) va letto come indice SQL, quindi non è il secondo come sarebbe in Java.

```
try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
        Statement stmt = conn.createStatement()) {
    ResultSet rs = stmt.executeQuery("SELECT first_name FROM coders ORDER BY 1");

    List<String> results = new ArrayList<String>();
    while (rs.next()) {
        results.add(rs.getString(1));
    }

    // …
}
```

il driver è la connessione col database, specificato nei parametri

executeQuery() on SELECT

itera sul result set

ArrayList=implementa l'interfaccia List, con i suoi metodi ma il tutto sarà messo in memoria come un array

legge la prima colonna della riga corrente del result set come stringa

resultset è un po' java e un po' sql, se la trasformo in arraylist è più leggibile in Java.

qui abbiamo la conversione in java puro di tutti gli elementi che mi arrivano da coders

SELECT via JDBC (prepared)

è come se fosse una query ma incompleta

```
/* String CODERS_BY_SALARY = "SELECT first_name, last_name, salary FROM coders "
            + "WHERE salary >= ? ORDER BY 3 DESC"; */

// …

try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD); //
        PreparedStatement prepStmt = conn.prepareStatement(CODERS_BY_SALARY)) {
    prepStmt.setDouble(1, lower);

    ResultSet rs = prepStmt.executeQuery();

    List<Coder> results = new ArrayList<>();
    while (rs.next()) {
        results.add(new Coder(rs.getString(1), rs.getString(2), rs.getInt(3)));
    }

    // …
}
```

tanti punti di domanda quanti sono i parametri che voglio mettere dentro

preparo la query

una volta che ho il prepared statement pronto posso cambiare il parametro quando voglio. in questo caso al posto del 1 punto di domanda metto lower

eseguo

Transazioni

finché non si fa la commit i cambiamenti sono visibili solo a me (in sql). in java, c'è autocommit di default. in jdbc quando apro la connection funziona allo stesso modo, se non voglio che ci sia l'autocommit gli passo il booleano false in modo che non si autocommitti. Se faccio questa cosa ovviamente devo ricordarmi di committare o fare il rollback.

- By default, una connessione è in modalità autocommit, ogni statement viene committato
- Connection.setAutoCommit(boolean)
- Connection.commit()
- Connection.rollback()

due linguaggi con logiche differenti, come li mettiamo in relazione?

Oggetti – Relazioni

- Java è un linguaggio Object-Oriented
  - Gestione della logica dell'applicazione, gestisce i dati usando
    - Ereditarietà
    - Accesso a dati correlati via collezioni o relazioni *has-a*
    - Tipi di dato propri del linguaggio (int, String, …)
- MySQL è un DBMS relazionale
  - Modella i dati usati dall'applicazione, gestisce i dati usando
    - Tipi di dato SQL (VARCHAR, DECIMAL, …)
    - JOIN, Primary Key, Foreign Key

Operazioni CRUD

- Create: inserimento di un oggetto/record nel database via INSERT
- Read: lettura di un record/oggetto via SELECT
  - Per chiave sono sicura che torni una riga sola
  - Per attributo
    - Potrebbe ritornare una collezione di oggetti
- Update: aggiornamento di un record/oggetto via UPDATE
- Delete: eliminazione di un record via DELETE

Data Access Object (DAO)

mvc è un pattern che riguarda l'architettura in generale. il dao lo usiamo per facilitare l'accesso a una sorgente, che sia database o altro. cioè pattern è come se fossero delle istruzioni per svolgere attività

- Pattern per semplificare l’accesso a data source
  - DAL: Data Access Layer
- Problema
  - L’implementazione delle funzionalità CRUD (Create Read Update Delete) dipende dal DBMS utilizzato
- Soluzione
  - La logica di accesso ai dati è isolata nell’oggetto DAO così se dobbiamo cambiare qualcosa o ricercare un errore sappiamo dove trovarlo
- Conseguenze
  - Miglior portabilità dell’applicazione per diversi DBMS
  - Il codice all’esterno del DAO è indipendente dal data souce

DTO e DAO

- Data Transfer Object
  - JavaBean ad esempio il nostro coder, è l'oggetto che fa da connettore tra java e sql
  - Valori scambiati tra database e applicazione
- Data Access Object
  - Interfaccia che dichiara le operazioni disponibili (CRUD et al.)
- Oggetto DAO
  - Implementa l’interfaccia per un particolare database
  - Tipicamente istanziato via factory method