JDBC (permette di usare DBMS con java) è un framework.
Differenza tra framework e library:
- **library** è un'insieme di classi, oggetti etc che da soli non fanno niente (bisogna usarli).
- **framework** è una struttura con un buco; riempiendo il buco, lancio il programma e il framework chiama i metodi di chi lo usa (il framework è più facile da riutilizzare).

ECLIPSE (classic o for java...)
Ci serve un DB; useremo un DB facile a cui accedere (da scaricare dal sito)
www.apache.org → cerca derby (DB)
(http://db.apache.org/derby/derby_downloads.html)

Per lanciarlo, scegliamo dove mettere il DB e lo lanciamo con una linea di comando.
- noSecurityManager: la sicurezza si sposta sulla rete piuttosto che sul DBMS.

SHELL
ij versione 10.10
// apro la connessione per poterci poi lavorare con Eclipse:
ij> connect 'jdbc:derby://localhost//Users/Aldo/Ateneo;create=true';
// jdbc: postgres.
// localhost: dialogo con il DBMS non con derby o altro.
// create = true: lo crea se non esiste.
// tutto il DB starà dentro Ateneo.
ij> create table studenti(matricola **int** generated always as identity, cognome varchar(50), nome varchar(50)); // Creo una tabella;
// Non esiste standard per l'autoincremento quindi bisogna vedere come fa il DB che si sta usando
ij> insert into studenti(cognome, nome) values ('Rossi','Giovanni');
ij> insert into studenti(cognome, nome) values ('Verdi','Giuseppe');
ij> insert into studenti(cognome, nome) values ('Bianchi','Mario');
ij> select * from studenti;
... stampa dati
// Quando finisco di usarlo, termino la connessione:
ij> disconnect;

ECLIPSE
- creo un nuovo progetto "Esercizio1" scegliendo il JRE di **default** e la **default** location; cambio la prospective java, se lo chiede.
- in src (voglio fare una cosa univoca e gerarchica con i vari nomi dell'unità organizzativa per cui lavoro) creo quindi un package con nome "it.unipr.informatica.Esercizio1"
- nel **package** creo un java **class** chiamandola "Esercizio1.java" e in essa mettiamo il main
- costruiamo una cartella nel progetto chiamata libs in cui si mettono tutte le librerie
- in libs mettiamo una libreria java (scaricata con derby che si chiama "derbyclient.jar" da copiare e incollare in libs.
- clicco col destro su derbyclient → build path

```
//Esercizio1.java
package it.unipr.informatica.esercizio1;

import java.sql.Connection;
import java.sql.DriverManager;
```

```
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;

public class Esercizio1 {
  public class static void main(String[] args) {
    try {
        // voglio caricare la classe:
-       new org.apache.derby.jdbc.ClientDriver(); // creando un oggetto di una classe, la classe stessa si carica
        Class.forName("org.apache.derby.jdbc.ClientDriver"); //oppure carico esplicitamente la classe senza creare oggetti
        // Con questo secondo metodo, posso prendere la stringa tramite file ed usare lo stesso codice per altre situazioni (siamo svincolati dal nome della classe)
        // Attenzione però che così si genera overhead quindi meglio non esagerare

        //Quando il Driver è attivo posso collegarmi
        Connection connection = DriverManager.getConnection("jbdc:derby://localhost/Ateneo");
        //Tutte le chiamate di questo tipo che falliscono restituiscono eccezione (e non è giustissimo perché l'eccezione non si risolve)
-       Statement statement = connection.createStatement();
-       ResultSet resultSet = statement.executeQuery("SELECT * FROM STUDENTI");  // resultSet è un iteratore che scandisce la tabella data dalla SELECT
        // executeQuery è specifica per le query
        // Quella SELECT non è parametrica; se volessi tutti i nomi che iniziano con r dovrei fare:
-       ResultSet resultSet = statement.executeQuery("SELECT * FROM STUDENTI
                        WHERE COGNOME LIKE '" + iniziale +"%'"); // ma ho dei problemi:
        // - il % non è standard
        // - tutte le volte il driver fa il parsing dell'SQL, anche se cambia solo la lettera (e questo rallenta).
        // Quindi risolviamo:
-       PreparedStatement statement = connection.preparedStatement("SELECT * FROM STUDENTI
                        WHERE COGNOME LIKE ?");  //il ? indica che aggiungerò qualcosa dopo
-       String iniziale = "R";
-       statement.setString(1,iniziale + "%");
-       ResultSet resultSet = statement.executeQuery();
        // Queste 3 righe sono più efficienti e non devo riportare in modo esatto il valore del parametro. Il ? sono dei valori quindi non si possono mettere parti di statement (questo protegge da attacchi di hacker). Problema del SQL ijection (che in questo caso non c'è).
-       statement.close();

// INSERIMENTO
// Vorrò ottenere nella stampa anche il valore di un contatore automatico (la matricola); per fare ciò devo:
// - mettere "Statement.RETURN_GENERATED_KEYS" come parametro quando creo lo Statement
// - uso "getGeneratedKeys()" dello statement per ottenere il valore del contatore automatico (se ne ho più di uno, otterrò una lista di valori).
Quindi "getGeneratedKeys()" ritorna un ResultSet di una riga e un numero di colonne pari al numero dei contatori automatici.
// - nel nostro caso, basta prendere la prima colonna
("chiavi.next();    int matricolaGenerata = chiavi.getInt(1)")
        PreparedStatement statement = connection.prepareStatement("INSERT INTO STUDENTI
                (COGNOME, NOME) VALUES (?, ?)", Statement.RETURN_GENERATED_KEYS);
```

```
        statement.setString(1,"Gialli");  // 1 indica che sto andando a modificare il 1° “?”
        statement.setString(2,"Grigi");  // 2 indica che sto andando a modificare il 2° “?”
        statement.execute();  // Preparato lo statement, lo eseguo
        ResultSet chiavi = statement.getGeneratedKeys();  // Genera la matricola
        chiavi.next();
        int matricolaGenerata = chiavi.getInt(1);  // 1 è l'indice della colonna (gli indici partono da 1)
        chiavi.close();
        System.out.println("Aggiunto nuovo studente con matricola: " + matricolaGenerata);
        statement.close();
// SELEZIONE (stampa)
        statement = connection.prepareStatement("SELECT * FROM STUDENTI
                                                                WHERE COGNOME LIKE ?");
        String iniziale = "G";
        statement.setString(1, iniziale + "%");
        ResultSet resultSet = statement.executeQuery();

        while(resultSet.next()) {
                int matricola = resultSet.getInt("MATRICOLA");
                String cognome = resultSet.getString("COGNOME");
                String nome = resultSet.getString("NOME");
                System.out.println(matricola + ", " + cognome + ", " + nome);
        }

        resultSet.close();
        statement.close();

// CANCELLAZIONE
        statement = connection.prepareStatement("DELETE FROM STUDENTI
                                                                WHERE COGNOME = ?");
        statement.setString(1, "Grigi");
        statement.execute();
        statement.close();

        connection.commit(); //scrive definitivamente le modifiche
        connection.rollback(); //ritorna alla creazione iniziale o all'ultima commit

        connection.close();

    } catch(Throwable throwable) {  // Trova tutti gli errori
        throwable.printStackTrace(); // Dice la riga dell'errore
    }  // Essendo tutto in try catch non sappiamo dove avviene l'errore esattamente; e il programma
termina.
        // Con il webserver devo essere io a gestire la terminazione mettendo 3 try catch per ogni ".close"
finale (per assicurarmi che funzionino).
    }
}
```