

# UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA



Corso di Laurea Magistrale in Informatica

## Report Progetto Software Dependability

Studente:

**Andrea BUCCI**

**Mat. 0522501969**

ANNO ACCADEMICO 2024/2025

---

# CONTENTS

<b>List of Figures</b>	<b>6</b>
<b>List of Tables</b>	<b>7</b>
<b>1 Il primo task</b>	<b>8</b>
1.1 Set-up del progetto . . . . .	8
1.2 Sessione di correzione delle Issues . . . . .	9
1.2.1 Issue #1 . . . . .	10
1.2.2 Issue #2 . . . . .	11
1.2.3 Issue #3 . . . . .	12
1.2.4 Issue #4, #5, #6 . . . . .	12
1.2.5 Issue #7 . . . . .	13
1.2.6 Issue #8, #9 . . . . .	14
1.2.7 Issue #10 . . . . .	14
1.2.8 Issue #11 . . . . .	15
1.2.9 Issue #12, #13 . . . . .	16
1.2.10 Issue #14, #15 . . . . .	16
1.2.11 Issue #16 . . . . .	16
1.2.12 Issue #17, #18, #19, #20 . . . . .	17
1.2.13 Issue #21 . . . . .	17

## CONTENTS

---

1.2.14	Issue #22 . . . . .	19
1.2.15	Issue #23, #24, #25, #26, #27 . . . . .	20
<b>2</b>	<b>Il secondo task</b>	<b>21</b>
2.1	Utilizzo di Docker . . . . .	21
2.1.1	Dockerizzazione del mio progetto . . . . .	21
<b>3</b>	<b>Il terzo task</b>	<b>24</b>
3.1	Creazione di una web page tramite l'utilizzo della mia versione di DbUtils . . . . .	24
3.1.1	Creazione del progetto . . . . .	24
3.1.2	Implementazione . . . . .	25
<b>4</b>	<b>Il quarto task</b>	<b>27</b>
4.1	Code Coverage e Mutation Testing del progetto con JaCoCo e PiTest . . . . .	27
4.1.1	Utilizzo di JaCoCo . . . . .	28
4.1.2	Utilizzo di PiTest . . . . .	30
<b>5</b>	<b>Il quinto task</b>	<b>32</b>
5.1	Utilizzo di Randoop e Github Copilot per migliorare la code coverage del progetto . . . . .	32
5.1.1	Randoop e JaCoCo . . . . .	32
5.1.2	Randoop e PiTest . . . . .	35
5.1.3	Github Copilot e JaCoCo . . . . .	36
5.1.4	Github Copilot e PiTest . . . . .	37
<b>6</b>	<b>Il sesto task</b>	<b>40</b>
6.1	Analisi della sicurezza e vulnerabilità del progetto . . . . .	40
6.1.1	FindSecBugs . . . . .	40
6.1.2	Dependency-Check . . . . .	46
<b>7</b>	<b>Il settimo task</b>	<b>47</b>
7.1	Java Microbenchmark Harness . . . . .	47

---

## CONTENTS

---

7.1.1	Microbenchmarking su AsyncQueryRunner . . . . .	47
-------	-------------------------------------------------	----

---

## LIST OF FIGURES

1.1	Esito della prima build . . . . .	8
1.2	Esito della prima Java CI . . . . .	9
1.3	Report della repository . . . . .	9
1.4	Issues segnalate da SonarCloud . . . . .	10
1.5	Prima issue . . . . .	10
1.6	Seconda issue . . . . .	11
1.7	Terza issue . . . . .	12
1.8	Quarta, quinta e sesta issue . . . . .	12
1.9	Classe contenente le stringhe costanti di identificazione SQLException . . . . .	13
1.10	Settima issue . . . . .	13
1.11	Test corretto della settimana issue . . . . .	14
1.12	Decima issue . . . . .	14
1.13	Undicesima issue . . . . .	15
1.14	Soluzione all'undicesima issue . . . . .	15
1.15	Quattordicesima issue . . . . .	16
1.16	Sedicesima issue . . . . .	16
1.17	Diciassettesima issue . . . . .	17
1.18	Ventunesima issue . . . . .	17
1.19	Soluzione alla ventunesima issue . . . . .	18

## LIST OF FIGURES

---

1.20	Ventiduesima issue . . . . .	19
1.21	Soluzione ventiduesima issue . . . . .	19
1.22	Ventitreesima issue . . . . .	20
2.1	Il mio Dockerfile . . . . .	22
2.2	Docker Container . . . . .	22
2.3	La mia MainClass . . . . .	23
3.1	Dipendenza del mio progetto . . . . .	25
3.2	Dockerfile . . . . .	25
3.3	La pagina web con la tabella di pokèmon randomici . . . . .	26
4.1	Dipendenza di JaCoCo . . . . .	28
4.2	Report copertura del codice con JaCoCo . . . . .	28
4.3	Coverage del package dbutils . . . . .	29
4.4	Report di PiTest . . . . .	30
4.5	Il package dbutils nel report PiTest . . . . .	31
5.1	Le classi con cui Randoop lavorerà . . . . .	33
5.2	Un test di poche righe generato da Randoop . . . . .	33
5.3	Un test più complesso generato da Randoop . . . . .	34
5.4	Code coverage post Randoop . . . . .	34
5.5	Code coverage del package Dbutils post Randoop . . . . .	35
5.6	Mutation coverage del package Dbutils post Randoop . . . . .	36
5.7	Code coverage del progetto post Github-Copilot . . . . .	37
5.8	Code coverage del package dbutils post Github-Copilot . . . . .	37
5.9	Mutation coverage del package dbutils post Github-Copilot . . . . .	38
6.1	Report FindSecBugs . . . . .	41
6.2	Potenziale vulnerabilità sugli errori . . . . .	42
6.3	Potenziale SQL Injection . . . . .	42
6.4	Potenziale SQL Injection . . . . .	42
6.5	Soluzione alla SQL Injection . . . . .	43
6.6	Possibile SQL Injection quando non ci sono parametri . . . . .	44

## LIST OF FIGURES

---

6.7	Soluzione proposta . . . . .	45
6.8	Report finale di FindSecsBugs . . . . .	45
6.9	Report di Dependency-Check . . . . .	46
7.1	. . . . .	47
7.2	. . . . .	48

---

## LIST OF TABLES



---

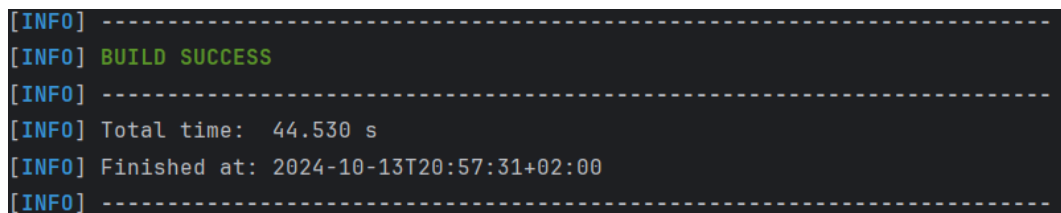
# CHAPTER 1

---

## IL PRIMO TASK

### 1.1 Set-up del progetto

La prima fase del progetto è stata quella di trovare un progetto Java abbastanza complesso da analizzare e dalla lista dei progetti Apache ho scelto *Apache Commons-DbUtils*. Dopo aver creato una fork sul mio profilo GitHub ho clonato la repository sul mio pc e successivamente ho fatto la build del progetto tramite il comando `mvn` che ha avuto esito positivo.

A terminal window with a dark background showing the output of a Maven build. The text is as follows:

```
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 44.530 s  
[INFO] Finished at: 2024-10-13T20:57:31+02:00  
[INFO] -----
```

Figure 1.1: Esito della prima build

Da GitHub è stata eseguita un'action `Java CI` che ha dato esito positivo per la prima, mentre l'altra è fallita a causa della versione early-access di Java 24 che è stata poi disabilitata nel `maven.yaml`

## 1. IL PRIMO TASK

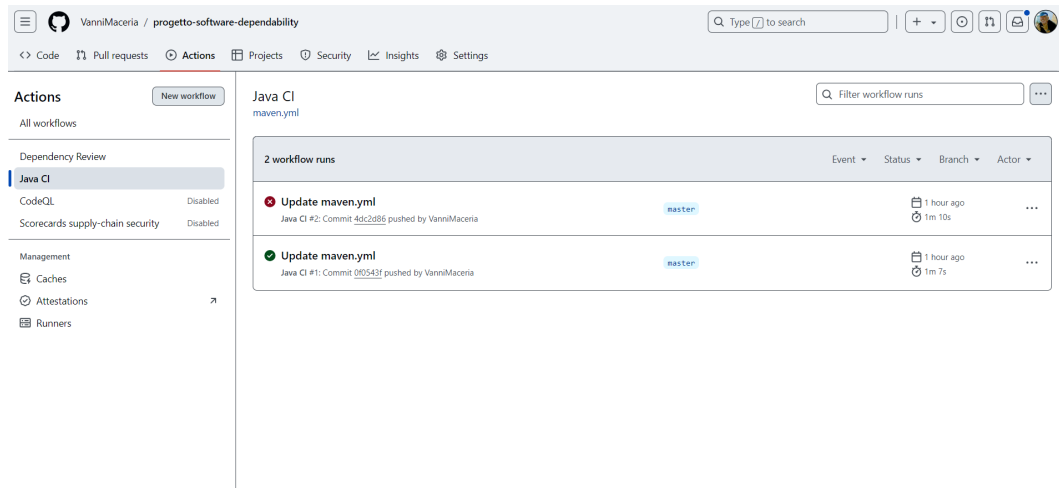


Figure 1.2: Esito della prima Java CI

Il passo successivo è stato quello di usare *SonarCloud*, tool SaaS (Software as a Service) usato per l'analisi del codice che ha riscontrato i problemi visibili nell'immagine.

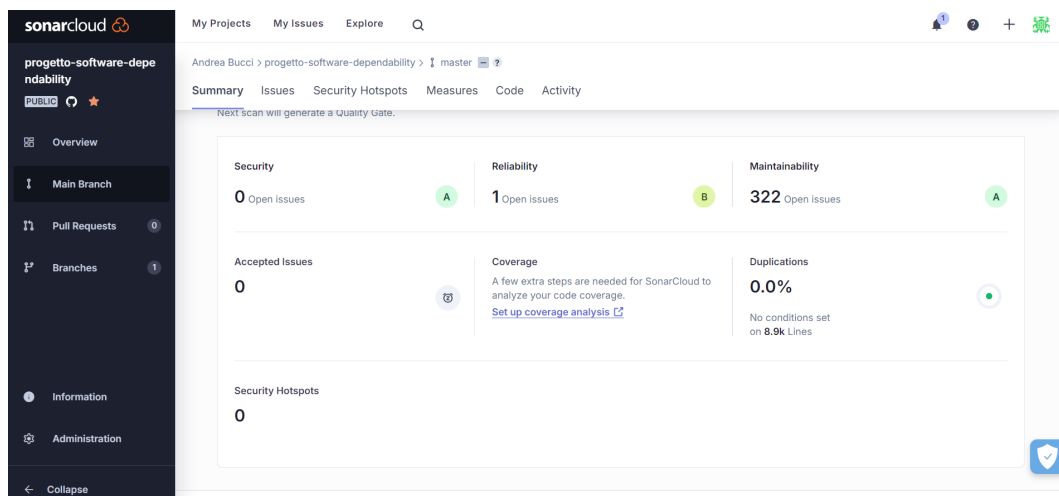


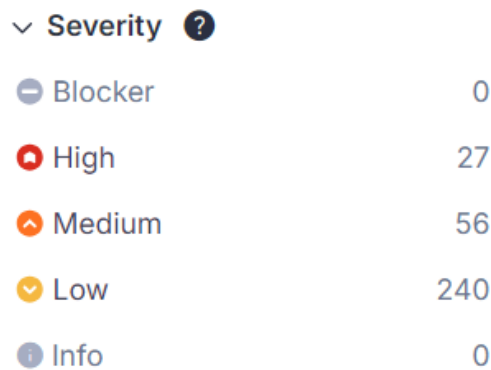
Figure 1.3: Report della repository

## 1.2 Sessione di correzione delle Issues

In questa sezione analizzeremo il modo in cui si è risolto (o si è tentato di risolvere) le funzioni o porzioni di codice che SonarCloud contrassegnava come BLOCKER e HIGH.

## 1. IL PRIMO TASK

---



A screenshot of the SonarCloud interface showing a dropdown menu for 'Severity' with a question mark icon. The menu lists five categories with their respective counts: Blocker (0), High (27), Medium (56), Low (240), and Info (0). Each category is preceded by a small circular icon representing its severity level.

Severity	Count
Blocker	0
High	27
Medium	56
Low	240
Info	0

Figure 1.4: Issues segnalate da SonarCloud

### 1.2.1 Issue #1

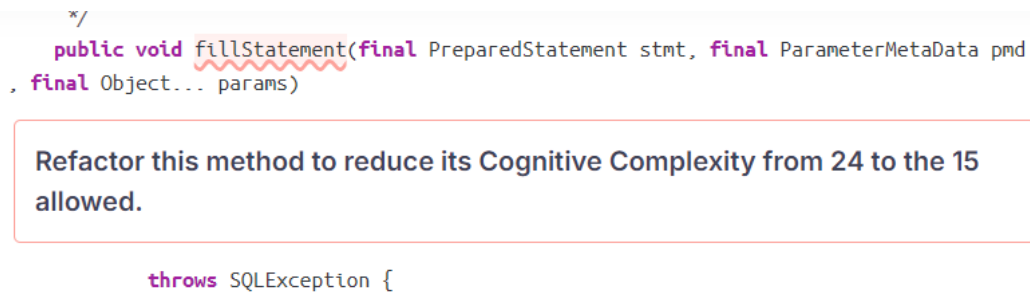


Figure 1.5: Prima issue

La soluzione che ho tentato per ridurre la complessità cognitiva del metodo è stata quella di suddividerlo in metodi più piccoli, ciascuno responsabile di un compito specifico.

Inizialmente, il metodo originale `fillStatement` faceva troppe cose contemporaneamente: controllava che il numero di parametri passati fosse corretto, gestiva i parametri di un `PreparedStatement` o un `CallableStatement`, e si occupava anche dei parametri nulli. Tutto questo rendeva il metodo complesso e più difficile da comprendere e mantenere.

La prima modifica che ho fatto è stata separare il controllo sul numero dei parametri in un metodo a parte, `checkParameterCount`. In questo modo, il controllo di corrispondenza tra il numero di parametri attesi e quelli effettivamente passati è isolato, rendendo più chiaro questo passaggio.

---

## 1. IL PRIMO TASK

---

Successivamente, ho estratto la logica che verifica se lo statement è un `CallableStatement` in un altro metodo dedicato, `getCallableStatement`. Anche questa parte era una piccola ma significativa responsabilità che appesantiva il metodo principale, e suddividerla ha semplificato la comprensione del flusso generale.

La parte più complessa riguardava la gestione dei parametri, sia quando erano nulli che quando non lo erano. Anche questa l'ho suddivisa in due metodi distinti: `handleNonNullParameter` per gestire i parametri che non sono nulli, e `handleNullParameter` per quelli che invece lo sono. Questo permette di gestire separatamente le due situazioni, rendendo la logica molto più semplice e lineare.

Il metodo principale, `fillStatement`, è così diventato molto più leggibile. Ora, anziché fare tutto da solo, delega le singole operazioni a metodi più piccoli e specifici.

### 1.2.2 Issue #2

```
protected int[] mapColumnsToProperties(final ResultSetMetaData rsmd,
```

**Refactor this method to reduce its Cognitive Complexity from 21 to the 15 allowed.**

```
final PropertyDescriptor[] props) throws SQLException {
```

Figure 1.6: Seconda issue

Anche in questo caso per ridurre la complessità cognitiva ho dovuto suddividere il metodo `mapColumnsToProperties` in più metodi che si occupano di compiti specifici. Sostanzialmente:

- `getColumnName` ha il compito di ottenere il nome di una colonna;
- `mapColumnToProperty` gestisce la mappatura tra colonna e proprietà;
- `resolvePropertyName` risolve il nome della proprietà usando eventuali

---

## 1. IL PRIMO TASK

---

override predefiniti o il nome della colonna. Se nessuno è disponibile, usa l'indice della colonna come ripiego;

- `getPropertyColumnName` si occupa di ottenere il nome della proprietà, gestendo sia la presenza di annotazioni `@Column` che il nome della proprietà stessa.

### 1.2.3 Issue #3

```
public GenerousBeanProcessor() {
```

Add a nested comment explaining why this method is empty, throw an `UnsupportedOperationException` or complete the implementation.

```
}
```

Figure 1.7: Terza issue

Banalmente ho aggiunto un commento che spiega che il costruttore è stato lasciato vuoto intenzionalmente.

### 1.2.4 Issue #4, #5, #6

```
public int[] batch(final Connection conn, final String sql, final Object[][] params
) throws SQLException {
    if (conn == null) {
        throw new SQLException("Null connection");
    }
}
```

Define a constant instead of duplicating this literal "Null connection" 7 times.

Figure 1.8: Quarta, quinta e sesta issue

In questo caso risolvere l'issue è stato molto semplice. Tutto ciò che ho fatto è stato creare una classe `SQLExceptionConstants` che definisce delle stringhe costanti da sostituire a quelle presenti in `SQLException(String error)`.

---

## 1. IL PRIMO TASK

---

```
1 package org.apache.commons.dbutils.constants;
2
3 public class SQLExceptionConstants { 20 usages
4     public static final String NULL_CONNECTION_ERROR = "Null connection error"; 7 usages
5     public static final String NULL_PARAMS_ERROR = "Null parameters error"; 2 usages
6     public static final String NULL_STATEMENT_ERROR = "Null SQL statement"; 7 usages
7     public static final String NULL_RESULT_SET_ERROR = "Null result set error"; 3 usages
8 }
```

Figure 1.9: Classe contenente le stringhe costanti di identificazione SQLException

### 1.2.5 Issue #7

```
@Test
public void testCloseNullConnection() throws Exception {
```

Add at least one assertion to this test case.

```
    DbUtils.close((Connection) null);
}
```

---

Figure 1.10: Settima issue

La settima issue nasce dalla mancanza di almeno un **Assert**, meccanismo di JUnit utilizzato per verificare che il risultato di un'operazione o l'output di un metodo corrisponda al comportamento atteso durante un test. In altre parole, il test esegue un'operazione `DbUtils.close((Connection) null)`, ma non contiene alcuna asserzione per confermare che il metodo si comporti come previsto. Come prima cosa ho inizializzato una variabile booleana `throwsException` a `false`, poi in un blocco `try-catch` ho eseguito la chiusura della connessione al database passando `null` castato a `Connection` come parametro. Nel caso venisse catturata un'eccezione `throwsException` diventa `true`. Infine chiamo `assertFalse` che verifica se effettivamente la variabile in questione abbia valore falso, se sì, il test viene superato, altrimenti fallirà mostrando la stringa passata come parametro.

---

## 1. IL PRIMO TASK

---

```
//issue #7
@Test  ▲ VanniMaceria +1
public void testCloseNullConnection() throws Exception {
    boolean throwsException = false;

    try {
        DbUtils.close((Connection) null);
    } catch (Exception e) {
        throwsException = true;
    }

    //mi aspetto che il valore di throwsException sia false, se così non è printa il messaggio
    assertFalse( message: "The value of SQL Connection can also be null", throwsException);
}
```

Figure 1.11: Test corretto della settima issue

### 1.2.6 Issue #8, #9

Come per l'issue #7 la soluzione è stata aggiungere un'Assert per i metodi `testCloseNullResultSet` e `testCloseNullStatement`.

### 1.2.7 Issue #10

```
@Test
public void testCloseQuietlyConnectionThrowingException() throws Exception {
```

**Add at least one assertion to this test case.**

```
    final Connection mockConnection = mock(Connection.class);
    doThrow(SQLException.class).when(mockConnection).close();
    DbUtils.closeQuietly(mockConnection);
}
```

Figure 1.12: Decima issue

Questo test verifica il comportamento del metodo `closeQuietly` quando viene chiamato su una connessione che lancia un'eccezione. Prima di tutto, viene creato un mock dell'oggetto `Connection` utilizzando Mockito, che permette di simulare una connessione a un database. Successivamente, il test configura il mock in modo che, quando viene invocato il metodo `close` su di esso, venga lanciata un'eccezione di tipo `SQLException`. A questo punto, il test chiama il metodo `DbUtils.closeQuietly()` passando il mock come

argomento.

Il metodo `closeQuietly()` tenta di chiudere la connessione, ma poiché il mock è configurato per lanciare un'eccezione, quest'ultima viene silenziata e non propagata. La soluzione che ho proposto è stata quella di usare `verify(mockConnection).close()` dove verifico che il metodo `close()` sia stato effettivamente chiamato sul mock.

### 1.2.8 Issue #11

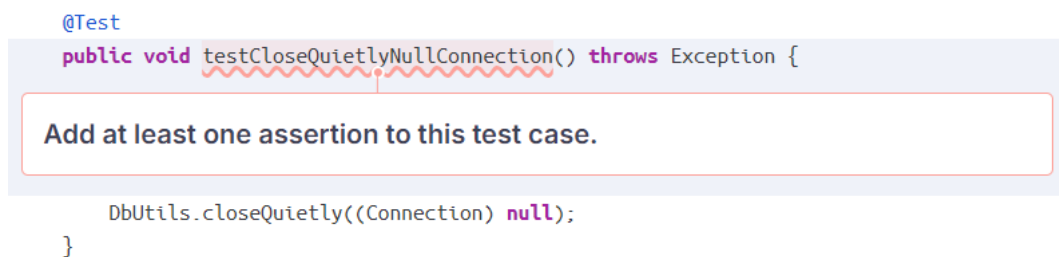


Figure 1.13: Undicesima issue

Il test verificava che il metodo `closeQuietly` non sollevasse eccezioni quando veniva chiamato con una connessione nulla. Inizialmente, non c'era alcuna asserzione, quindi il test semplicemente eseguiva il metodo e si aspettava che non fallisse. Per migliorarlo, è stata aggiunta l'asserzione `assertDoesNotThrow`, che controlla esplicitamente che il metodo non generi eccezioni quando viene passato un valore `null` per la `Connection`. Questo ha reso il test più robusto poiché ora viene verificato in modo formale il comportamento del codice.

```
//issue #11
@Test  * Gary Gregory *
public void testCloseQuietlyNullConnection() throws Exception {
    assertDoesNotThrow(() -> {
        DbUtils.closeQuietly((Connection) null);
    });
}
```

Figure 1.14: Soluzione all'undicesima issue



### 1.2.9 Issue #12, #13

Così come per la issue #11 la soluzione è stata quella di aggiungere un `assertDoesNotThrow`, che controlla che il metodo non generi eccezioni quando viene passato un valore `null` per il `ResultSet` e lo `Statement`.

### 1.2.10 Issue #14, #15

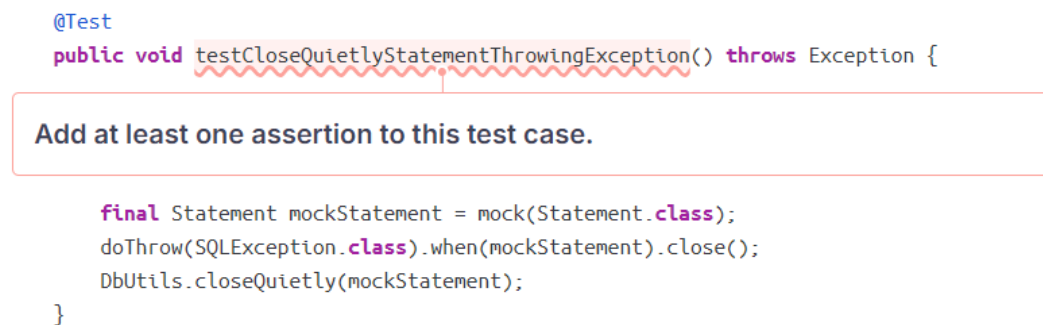


Figure 1.15: Quattordicesima issue

La soluzione alle issue #14 e #15 è pressochè identica a quella proposta nella issue #10, quello che cambia è il tipo del mock verificato.

### 1.2.11 Issue #16

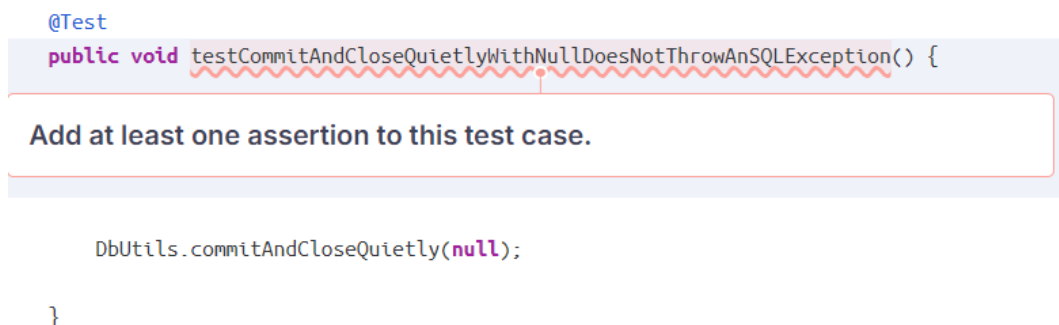


Figure 1.16: Sedicesima issue

La soluzione proposta è stata quella di usare l'asserzione `assertDoesNotThrow(() -> DbUtils.commitAndCloseQuietly(null));`

---

## 1. IL PRIMO TASK

---

che verifica che il metodo `commitAndCloseQuietly` venga eseguito senza lanciare eccezioni, anche se viene passato un valore `null`.

### 1.2.12 Issue #17, #18, #19, #20

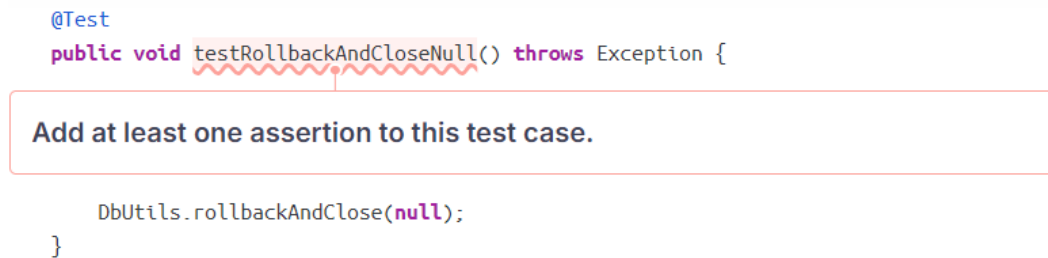


Figure 1.17: Diciassettesima issue

Dovendo verificare anche in questo test che il metodo chiamato non lancia eccezione, anche qui ho inserito `assertDoesNotThrow(() -> DbUtils.rollbackAndClose(null));`. Per il test #18 è stata fatta la stessa cosa con testando il metodo `DbUtils.rollbackAndCloseQuietly(null)`. Anche per i test #19 e #20 è stato eseguito lo stesso approccio tramite l'uso di `assertDoesNotThrow(() -> DbUtils.metodoDaTestare(null));`

### 1.2.13 Issue #21

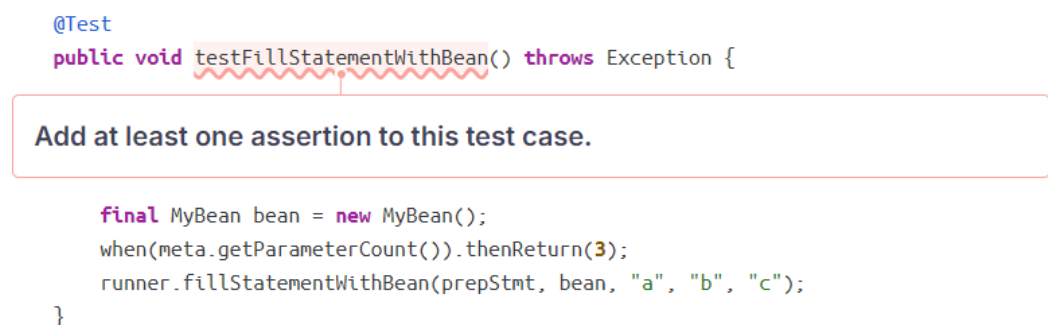


Figure 1.18: Ventunesima issue

---

## 1. IL PRIMO TASK

---

Il test originale verificava esplicitamente alcun comportamento del metodo. Il semplice fatto di chiamare il metodo `fillStatementWithBean` non garantiva che il metodo stesse funzionando correttamente. Mancava una verifica esplicita sul risultato dell'esecuzione del metodo o su eventuali effetti collaterali, come ad esempio se il `PreparedStatement` fosse stato riempito correttamente.

```
//issue #21
@Test  * Gary Gregory *
public void testFillStatementWithBean() throws Exception {
    final MyBean bean = new MyBean();
    bean.setA(123);
    bean.setB(20.00);
    bean.setC("stringa prova");

    when(meta.getParameterCount()).thenReturn(3);

    runner.fillStatementWithBean(preparedStatement, bean, ...propertyNames: "a", "b", "c");

    // Assert: Verifica che i metodi del PreparedStatement siano stati chiamati con i valori corretti
    verify(preparedStatement).setObject( parameterIndex: 1, x: 123);
    verify(preparedStatement).setObject( parameterIndex: 2, x: 20.00);
    verify(preparedStatement).setObject( parameterIndex: 3, x: "stringa prova");
}
```

Figure 1.19: Soluzione alla ventunesima issue

La soluzione è stata quella di aggiungere asserzioni utilizzando Mockito's `verify`. Facendo ciò ho verificato che il `PreparedStatement` fosse riempito con i valori corretti delle proprietà del bean, rispettando i tipi dei parametri. Ad esempio `verify(preparedStatement).setObject(1, 123)`: verifica che il primo parametro nel `PreparedStatement` sia stato impostato correttamente con il valore di `bean.getA()` che è un intero.

### 1.2.14 Issue #22

```
@Test
public void testCreatesResultSetIteratorTakingThreeArgumentsAndCallsRemove() {

    final ResultSet resultSet = mock(ResultSet.class);
    final ResultSetIterator resultSetIterator = new ResultSetIterator(resultSet, null
);
    resultSetIterator.remove();

}
```

**Add at least one assertion to this test case.**

Figure 1.20: Ventiduesima issue

Il test attuale chiama semplicemente il metodo `remove()` su un'istanza di `ResultSetIterator`, ma non verifica esplicitamente se il metodo funziona correttamente.

```
//issue #22
@Test  Gary Gregory *
public void testCreatesResultSetIteratorTakingThreeArgumentsAndCallsRemove() throws SQLException {
    final ResultSet resultSet = mock(ResultSet.class);
    final ResultSetIterator resultSetIterator = new ResultSetIterator(resultSet, convert: null);

    resultSetIterator.remove();

    // Assert: Verifica che il metodo deleteRow() del ResultSet sia stato chiamato
    verify(resultSet).deleteRow();
}
```

Figure 1.21: Soluzione ventiduesima issue

Ora il test utilizza `Mockito.verify()` per assicurarsi che `deleteRow()` sia stato effettivamente chiamato, confermando che il metodo `remove()` funziona come previsto. Se il metodo non fosse stato chiamato, il test fallirebbe, indicando un problema nel comportamento atteso.

### 1.2.15 Issue #23, #24, #25, #26, #27

```
/**  
 * Constructor for TestBean.  
 */  
public TestBean() {
```

Add a nested comment explaining why this method is empty, throw an `UnsupportedOperationException` or complete the implementation.

```
}
```

Figure 1.22: Ventitreesima issue

Per risolvere queste issues comuni tra loro è bastato inserire un commento all'interno del costruttore o del metodo richiesto.

---

---

## CHAPTER 2

---

### IL SECONDO TASK

#### 2.1 Utilizzo di Docker

Il secondo task ha riguardato l'utilizzo di Docker. Docker è una piattaforma che consente di creare, distribuire e gestire applicazioni in container. Un container è un'unità leggera e portatile che include tutto il necessario per eseguire un'applicazione: il codice, le librerie, le dipendenze e il sistema operativo, in modo isolato.

##### 2.1.1 Dockerizzazione del mio progetto

Successivamente all'installazione di Docker, il primo passo è stato creare il `.jar` del progetto tramite il comando `mvn install`.

Il perno del processo è stato scrivere il `Dockerfile`, un file contenente delle istruzioni che Docker usa per costruire un'immagine Docker.

## 2. IL SECONDO TASK

---

```
# Usa un'immagine base con OpenJDK 11
FROM openjdk:11

# Copia il file JAR nella directory di lavoro
ADD target/commons-dbutils-1.9.0-SNAPSHOT.jar commons-dbutils.jar

# Espone la porta 8080
EXPOSE 8080

# Comando per eseguire l'applicazione
ENTRYPOINT ["java", "-jar", "commons-dbutils.jar"]
```

Figure 2.1: Il mio Dockerfile

Infine ho buildato l'immagine Docker con il comando `docker build -t commons-dbutils .` e runnando quest'ultima si è ottenuto il funzionamento del progetto in un container Docker.

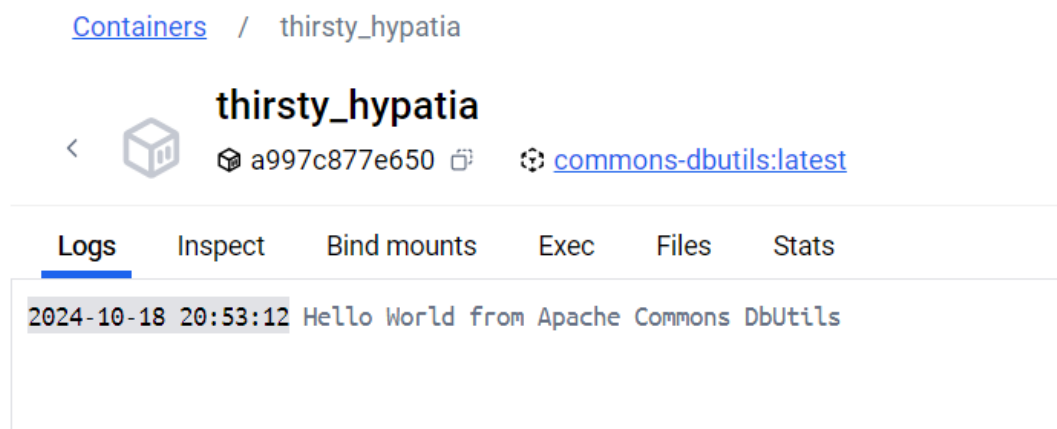


Figure 2.2: Docker Container

**\*NOTA\*:** In un primo momento il tentativo di runnare l'immagine veniva interrotto immediatamente a causa della mancanza di una classe che contenesse un `main()`, perciò ho creato la più classica delle classi: "HelloWorld".

## 2. IL SECONDO TASK

---

```
package org.apache.commons.dbutils;  
⚡  
public final class MainClass {  ⚡ VanniMaceria  
  
    private MainClass() {  no usages  ⚡ VanniMaceria  
  
    }  
  
    public static void main(String[] args) {  ⚡ VanniMaceria  
        System.out.println("Hello World from Apache Commons DbUtils");  
    }  
}
```

Figure 2.3: La mia MainClass



---

---

## CHAPTER 3

---

### IL TERZO TASK

#### 3.1 Creazione di una web page tramite l'utilizzo della mia versione di DbUtils

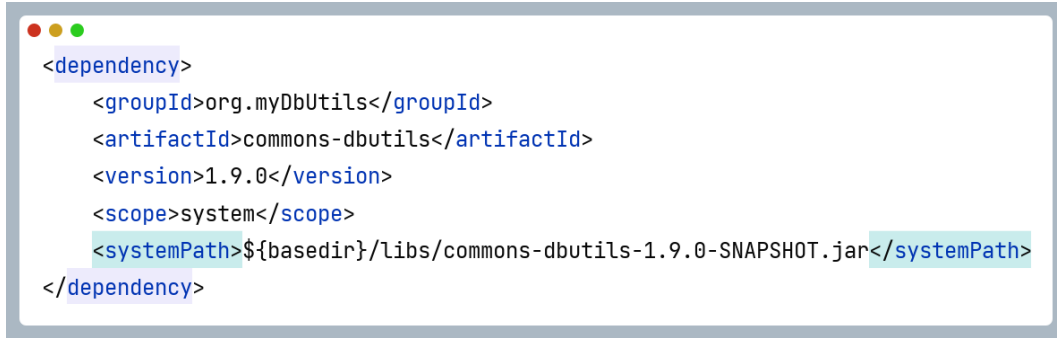
##### 3.1.1 Creazione del progetto

Come prima cosa è stato creato un progetto Java ed è stata aggiunta la dipendenza della mia versione di DbUtils. Ciò è stato fatto aggiungendo al `pom.xml` il `.jar` del mio DbUtils. Successivamente è stato aggiunto anche il driver di JDBC per poter interfacciare il progetto con il mio database MySQL. La repository è disponibile a <https://github.com/VanniMaceria/WebPageSoftwareDependability>

---

### 3. IL TERZO TASK

---



```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>commons-dbutils</artifactId>
  <version>1.9.0</version>
  <scope>system</scope>
  <systemPath>${basedir}/libs/commons-dbutils-1.9.0-SNAPSHOT.jar</systemPath>
</dependency>
```

Figure 3.1: Dipendenza del mio progetto

#### 3.1.2 Implementazione

La prima classe che ho implementato è stata `PokemonBean`, che rappresenta un'oggetto "Pokémon", cioè l'entità che popola il mio database. La classe contiene i campi principali analoghi a quelli della tabella SQL. Di conseguenza ho creato getter, setter e `toString`.

Successivamente sono passato all'implementazione della classe `DBHandler`

Questa contiene i metodi per la connessione al database e per l'esecuzione della query che seleziona 3 pokémon randomicamente.

La classe che si occupa di gestire la richiesta `GET` all'endpoint `http://localhost:8000/pokemon` e di predisporre i dati in formato `HTML` per la risposta è `PokemonController`

Successivamente c'è la classe `App` che dal `main` inizializza un Web Server che gestisce le richieste `http`.

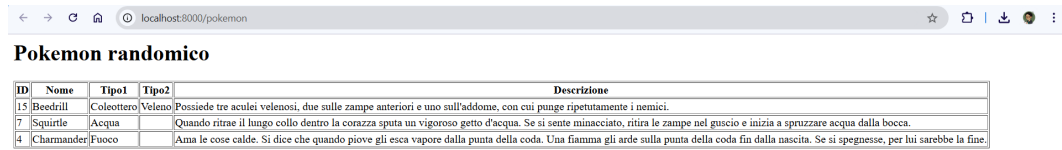
Infine il `Dockerfile` si occupa di creare l'immagine del progetto

```
FROM openjdk:17
ADD target/WebPageSwDependability-1.0-SNAPSHOT-jar-with-dependencies.jar web_page_sw_dependability.jar
ENTRYPOINT ["java", "-jar", "web_page_sw_dependability.jar"]
EXPOSE 8080
```

Figure 3.2: Dockerfile

Runnando il container Docker, all'indirizzo `http://localhost:8000/pokemon` comparirà la seguente pagina web:

### 3. IL TERZO TASK



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/pokemon'. The page title is 'Pokemon randomico'. Below the title is a table with four columns: ID, Nome, Tipo1, and Tipo2. The table contains three rows of data for different Pokémon: Beedrill, Squirtle, and Charmander. Each row includes a description of the Pokémon's abilities and characteristics.

ID	Nome	Tipo1	Tipo2	Descrizione
15	Beedrill	Coleottero	Veleno	Possiede tre aculei velenosi, due sulle zampe anteriori e uno sull'addome, con cui punge ripetutamente i nemici.
7	Squirtle	Acqua		Quando ritrae il lungo collo dentro la corazza sputa un vigoroso getto d'acqua. Se si sente minacciato, ritira le zampe nel guscio e inizia a spruzzare acqua dalla bocca.
4	Charmander	Fuoco		Ama le cose calde. Si dice che quando piove gli esca vapore dalla punta della coda. Una fiamma gli arde sulla punta della coda fin dalla nascita. Se si spegnesse, per lui sarebbe la fine.

Figure 3.3: La pagina web con la tabella di pokèmon randomici

---

---

## CHAPTER 4

---

### IL QUARTO TASK

#### 4.1 Code Coverage e Mutation Testing del progetto con JaCoCo e PiTest

In questo capitolo ho integrato JaCoCo e PiTest per analizzare la qualità del codice e l'efficacia dei test automatizzati all'interno della libreria Apache Commons DbUtils. JaCoCo è stato utilizzato per misurare la copertura del codice, fornendo dati sulle istruzioni, sui rami condizionali e sui metodi eseguiti dai test. Questo ha permesso di identificare parti del codice non testate e di migliorare la robustezza della suite di test. Parallelamente, PiTest è stato impiegato per effettuare il mutation testing, valutando la capacità dei test di individuare modifiche intenzionali nel codice (mutazioni). Questa combinazione di strumenti ha fornito un'analisi approfondita della qualità del progetto, evidenziando sia i punti di forza che le aree di miglioramento nel testing.

## 4. IL QUARTO TASK

### 4.1.1 Utilizzo di JaCoCo

Come primo step ho aggiunto la dipendenza di JaCoCo nel `pom.xml`. Successivamente, buildando il progetto con `mvn package`, al path `target/site/jacoco/index.html` viene generato un report in HTML.

```
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <executions>
    <execution>
      <phase>test</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Figure 4.1: Dipendenza di JaCoCo

Il report generato rappresenta la copertura del codice sorgente del mio progetto Apache Commons DbUtils. Questo file HTML mostra dettagli relativi a quali parti del codice sono state testate (cioè eseguite dai test unitari) e quali no. Analizzando la tabella possiamo dire che:

Apache Commons DbUtils

Apache Commons DbUtils

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.dbutils	<div><div></div></div> 59%		<div><div></div></div> 77%		276	562	469	1,129	225	428	4	20
org.apache.commons.dbutils.handlers	<div><div></div></div> 100%		<div><div></div></div> 100%		0	58	0	114	0	48	0	12
org.apache.commons.dbutils.wrappers	<div><div></div></div> 100%		<div><div></div></div> 93%		2	64	0	101	0	49	0	2
org.apache.commons.dbutils.handlers.columns	<div><div></div></div> 100%		<div><div></div></div> 82%		5	44	0	30	0	30	0	10
org.apache.commons.dbutils.handlers.properties	<div><div></div></div> 100%		<div><div></div></div> 94%		1	16	0	24	0	6	0	2
org.apache.commons.dbutils.constants	<div><div></div></div> 100%		n/a		0	1	0	2	0	1	0	1
Total	1,718 of 5,275	67%	68 of 364	81%	284	745	469	1,400	225	562	4	47

Figure 4.2: Report copertura del codice con JaCoCo

Per i package `handlers`, `wrappers`, `columns`, `properties` e `constants` si è ottenuta una **copertura delle istruzioni** del 100% il che indica l'esecuzione completa di tutte le istruzioni presenti in questi pacchetti, tuttavia, non riflette necessariamente l'efficacia dei test. Questo risultato serve più che altro a garantire che il codice scritto in tali componenti sia stato completamente testato dai test unitari. D'altra parte, il package principale, `org.apache.commons.dbutils`, presenta una copertura del 59%

## 4. IL QUARTO TASK

per le istruzioni e questo suggerisce che alcune parti fondamentali del codice non sono state testate. Analizzando le classi del package vediamo che

### org.apache.commons.dbutils

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
BaseResultSetHandler		2%		50%	188	193	283	291	187	192	0	1
QueryRunner		78%		82%	19	65	30	188	8	34	0	1
AbstractQueryRunner		80%		76%	20	62	39	174	6	32	0	1
DbUtils		47%		54%	10	31	31	82	5	20	0	1
AsyncQueryRunner.QueryCallableStatement		0%		0%	3	3	19	19	2	2	1	1
BeanProcessor		89%		85%	9	52	15	126	0	21	0	1
AsyncQueryRunner.BatchCallableStatement		0%		0%	3	3	15	15	2	2	1	1
AsyncQueryRunner.UpdateCallableStatement		0%		0%	3	3	15	15	2	2	1	1
AsyncQueryRunner		87%		n/a	5	41	4	32	5	41	0	1
ResultSetIterator		66%		100%	2	10	9	21	2	8	0	1
StatementConfiguration		82%		87%	2	22	1	23	0	14	0	1
OutParameter		78%		100%	3	10	3	21	3	9	0	1
BasicRowProcessor		94%		66%	4	15	1	28	1	9	0	1
QueryLoader		95%		83%	1	9	1	22	0	6	0	1
DbUtils.DriverProxy		89%		n/a	1	8	1	10	1	8	0	1
MainClass		0%		n/a	1	1	2	2	1	1	1	1
StatementConfiguration.Builder		98%		50%	1	9	0	14	0	8	0	1
GenerousBeanProcessor		100%		91%	1	8	0	19	0	2	0	1
BasicRowProcessor.CaseInsensitiveHashMap		100%		n/a	0	6	0	15	0	6	0	1
ProxyFactory		100%		n/a	0	11	0	12	0	11	0	1
Total	1.718 of 4.292	59%	60 of 268	77%	276	562	469	1.129	225	428	4	20

Figure 4.3: Coverage del package dbutils

le classi che fanno scendere la coverage sulle istruzioni sono `BaseResultSetHandler`, `DbUtils`, `AsyncQueryRunner.QueryCallableStatement`, `AsyncQueryRunner.BatchCallableStatement`,

`AsyncQueryRunner.UpdateCallableStatement`.

Per quanto concerne la **Branch Coverage** tutti i package hanno una buona copertura dei rami. Prendiamo ad esempio il package dbutils che presenta la copertura minore (77%). Osservando la figura 4.3 notiamo che le classi responsabili di ciò sono `BaseResultSetHandler`, `DbUtils`, `AsyncQueryRunner.QueryCallableStatement`, `AsyncQueryRunner.BatchCallableStatement`,

`AsyncQueryRunner.UpdateCallableStatement`, `StatementConfiguration.Builder`.

Il fatto che instruction coverage e branch coverage siano entrambe basse per le stesse classi indica che i test esistenti non coprono abbastanza percorsi condizionali e istruzioni all'interno del codice. La soluzione potrebbe essere identificare le aree non coperte, scrivere test mirati, e, se necessario, fare refactoring per semplificare il codice e migliorare la testabilità.

### 4.1.2 Utilizzo di PiTest

Dopo aver aggiunto il plugin PiTest, eseguendo il comando `mvn org.pitest:pitest-maven:mutationCoverage` si avvia il plugin per analizzare la copertura di mutazione, verificando se i test esistenti riescono a rilevare modifiche deliberate (mutazioni) nel codice. Alla fine del processo, al path `target/pit-reports` viene generato un report in formato HTML che se aperto ci comunica delle cose interessanti:

#### Pit Test Coverage Report

##### Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
40	67% <div><div></div></div> 959/1436	52% <div><div></div></div> 390/745	85% <div><div></div></div> 390/459

##### Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
<a href="#">org.apache.commons.dbutils</a>	14	59% <div><div></div></div> 690/1167	45% <div><div></div></div> 280/628	82% <div><div></div></div> 280/342
<a href="#">org.apache.commons.dbutils.handlers</a>	12	100% <div><div></div></div> 114/114	100% <div><div></div></div> 26/26	100% <div><div></div></div> 26/26
<a href="#">org.apache.commons.dbutils.handlers.columns</a>	10	100% <div><div></div></div> 30/30	87% <div><div></div></div> 33/38	87% <div><div></div></div> 33/38
<a href="#">org.apache.commons.dbutils.handlers.properties</a>	2	100% <div><div></div></div> 24/24	100% <div><div></div></div> 14/14	100% <div><div></div></div> 14/14
<a href="#">org.apache.commons.dbutils.wrappers</a>	2	100% <div><div></div></div> 101/101	95% <div><div></div></div> 37/39	95% <div><div></div></div> 37/39

Report generated by [PIT](#) 1.17.1

Enhanced functionality available at [arcmutate.com](#)

Figure 4.4: Report di PiTest

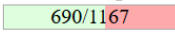
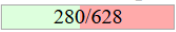
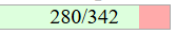
- **Line Coverage:** è il rapporto tra le istruzioni di riga coperte dai test e il numero totale di istruzioni di riga potenzialmente da eseguire nella classe;
- **Mutation Coverage:** è il rapporto tra il numero di mutazioni uccise dai test e il numero totale di mutazioni (indipendentemente dal fatto che siano state coperte o meno dai test);
- **Test Strength:** indica quanto sono efficaci i test nel rilevare le mutazioni nel codice.

Come nella code coverage, anche per il mutation testing il pacchetto che conviene analizzare è `dbutils`:

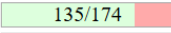
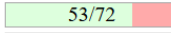
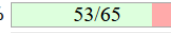
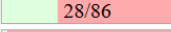
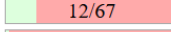
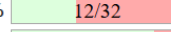
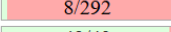
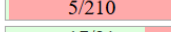
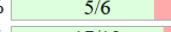
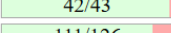
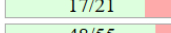
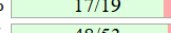
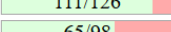
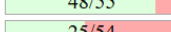
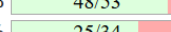
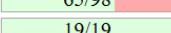
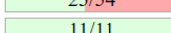
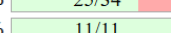
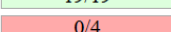
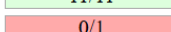
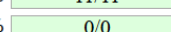
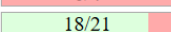
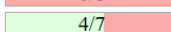
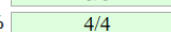
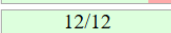
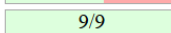
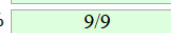
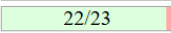
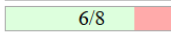
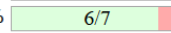
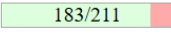
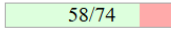
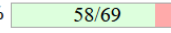
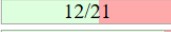
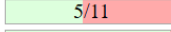
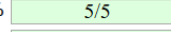
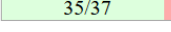
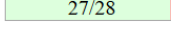
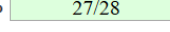



### Pit Test Coverage Report

#### Package Summary

org.apache.commons.dbutils

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
14	59% 	45% 	82% 

#### Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">AbstractQueryRunner.java</a>	78% 	74% 	82% 
<a href="#">AsyncQueryRunner.java</a>	33% 	18% 	38% 
<a href="#">BaseResultSetHandler.java</a>	3% 	2% 	83% 
<a href="#">BasicRowProcessor.java</a>	98% 	81% 	89% 
<a href="#">BeanProcessor.java</a>	88% 	87% 	91% 
<a href="#">DbUtils.java</a>	66% 	46% 	74% 
<a href="#">GenerousBeanProcessor.java</a>	100% 	100% 	100% 
<a href="#">MainClass.java</a>	0% 	0% 	100% 
<a href="#">OutParameter.java</a>	86% 	57% 	100% 
<a href="#">ProxyFactory.java</a>	100% 	100% 	100% 
<a href="#">QueryLoader.java</a>	96% 	75% 	86% 
<a href="#">QueryRunner.java</a>	87% 	78% 	84% 
<a href="#">ResultSetIterator.java</a>	57% 	45% 	100% 
<a href="#">StatementConfiguration.java</a>	95% 	96% 	96% 

Report generated by [PIT](#) 1.17.1

Figure 4.5: Il package dbutils nel report PiTest

Le classi che più richiedono un miglioramento sui casi di test sono: `AsyncQueryRunner`, `BaseResultSetHandler`, `DbUtils` e `ResultSetIterator.java`.

Nel prossimo capitolo utilizzerò i tool `Randoop` e `Github Copilot` per generare automaticamente dei casi di test, con l'obiettivo di aumentare gli attuali risultati ottenuti da `JaCoCo` e `PiTest`.



---

---

# CHAPTER 5

---

## IL QUINTO TASK

### 5.1 Utilizzo di Randoop e Github Copilot per migliorare la code coverage del progetto

Come detto nel capitolo precedente, migliorare la code coverage di un progetto è un'operazione delicata che richiede attenzione ai dettagli e soprattutto tempo. Tuttavia, l'adozione di strumenti di automazione come **Randoop** e **Github Copilot** consente di accelerare significativamente questo processo, generando delle volte, test efficaci e individuando potenziali lacune nel codice.

#### 5.1.1 Randoop e JaCoCo

Per generare test automatici con **Randoop** bisogna eseguire il comando

```
java -classpath ${RANDOOP_JAR} randoop.main.Main gentests  
--classlist=myclasses.txt --time-limit=60 dove:
```

- `${RANDOOP_JAR}` rappresenta il path dove è salvato il jar eseguibile di **Randoop**;

---

## 5. IL QUINTO TASK

---

- `myclasses.txt` è un file di testo contenente il nome delle classi su cui ho deciso di generare test automatici;
- `--time-limit=60` indica la modalità di generazione di test. In questo caso dopo 60 secondi Randoop smetterà di generare test.

Come visto precedentemente le classi che abbassavano la code coverage erano principalmente queste nella figura sottostante:

```
org.apache.commons.dbutils.AsyncQueryRunner
org.apache.commons.dbutils.BaseResultSetHandler
org.apache.commons.dbutils.DbUtils
org.apache.commons.dbutils.ResultSetIterator
org.apache.commons.dbutils.StatementConfiguration
```

Figure 5.1: Le classi con cui Randoop lavorerà

Una volta creato il file di testo con i nomi della classi da dare "in pasto" a Randoop, con il comando sopracitato parte la generazione dei casi di test. Alla fine dei 60 secondi sono stati generati quattro classi di test (`RegressionTest`) che presentano alcuni metodi di test semplici ed altri molto lunghi e complessi.

```
@Test
public void test0001() throws Throwable {
    if (debug)
        System.out.format("%n%s%n", "RegressionTest0.test0001");
    java.sql.Connection connection0 = null;
    org.apache.commons.dbutils.DbUtils.rollbackQuietly(connection0);
}
```

Figure 5.2: Un test di poche righe generato da Randoop

## 5. IL QUINTO TASK

```
@Test
public void test0020() throws Throwable {
    if (debug)
        System.out.format("%n%s%n", "RegressionTest0.test0020");
    java.util.concurrent.ExecutorService executorService0 = null;
    org.apache.commons.dbutils.QueryRunner queryRunner1 = null;
    org.apache.commons.dbutils.AsyncQueryRunner asyncQueryRunner2 = new org.apache.commons.dbutils.AsyncQueryRunner(executorService0, queryRunner1);
    java.sql.PreparedStatement preparedStatement3 = null;
    java.sql.ParameterMetaData parameterMetaData4 = null;
    java.sql.ResultSet resultSet6 = null;
    java.lang.Iterable<java.lang.Object[]> objArrayIterable7 = org.apache.commons.dbutils.ResultSetIterator.iterable(resultSet6);
    java.lang.Object[] objArray9 = new java.lang.Object[] { 100.0d, resultSet6, (byte) 0 };
    // The following exception was thrown during execution in test generation
    try {
        asyncQueryRunner2.fillStatement(preparedStatement3, parameterMetaData4, objArray9);
        org.junit.Assert.fail("Expected exception of type java.lang.NullPointerException; message: Cannot invoke \"java.sql.ResultSet.iterator()\" on null");
    } catch (java.lang.NullPointerException e) {
        // Expected exception.
    }
    org.junit.Assert.assertNotNull(objArrayIterable7);
    org.junit.Assert.assertNotNull(objArray9);
    org.junit.Assert.assertEquals(java.util.Arrays.deepToString(objArray9), actual: "[100.0, null, 0]");
    org.junit.Assert.assertEquals(java.util.Arrays.toString(objArray9), actual: "[100.0, null, 0]");
}
```

Figure 5.3: Un test più complesso generato da Randoop

Fatto ciò per ricalcolare la coverage è bastato rifare la build del progetto. Sfortunatamente, Randoop ha migliorato la instruction coverage solamente dell'1%, mentre la branch coverage è aumentata del 3%.

Apache Commons DbUtils

Apache Commons DbUtils

Element

Missed Instructions

Cov.

Missed Branches

Cov.

Missed

Cxty

Missed

Lines

Missed

Methods

Missed

Classes

org.apache.commons.dbutils	<div><div></div><div></div></div>	60%	<div><div></div><div></div></div>	80%	264	562	457	1.129	219	428	4	20		
org.apache.commons.dbutils.handlers	<div><div></div><div></div></div>	100%	<div><div></div><div></div></div>	100%	0	58	0	114	0	48	0	12		
org.apache.commons.dbutils.wrappers	<div><div></div><div></div></div>	100%	<div><div></div><div></div></div>	93%	2	64	0	101	0	49	0	2		
org.apache.commons.dbutils.handlers.columns	<div><div></div><div></div></div>	100%	<div><div></div><div></div></div>	82%	5	44	0	30	0	30	0	10		
org.apache.commons.dbutils.handlers.properties	<div><div></div><div></div></div>	100%	<div><div></div><div></div></div>	94%	1	16	0	24	0	6	0	2		
org.apache.commons.dbutils.constants	<div><div></div><div></div></div>	100%	<div><div></div><div></div></div>	n/a	0	1	0	2	0	1	0	1		
Total		1.683 of 5.275		68%	61 of 364	83%	272	745	457	1.400	219	562	4	47

Figure 5.4: Code coverage post Randoop

## 5. IL QUINTO TASK

### org.apache.commons.dbutils



































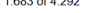





Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
BaseResultSetHandler		2%		50%	188	193	283	291	187	192	0	1
QueryRunner		78%		82%	19	65	30	188	8	34	0	1
AbstractQueryRunner		82%		85%	14	62	36	174	5	32	0	1
DbUtils		55%		59%	7	31	25	82	2	20	0	1
AsyncQueryRunner.QueryCallableStatement		0%		0%	3	3	19	19	2	2	1	1
BeanProcessor		89%		85%	9	52	15	126	0	21	0	1
AsyncQueryRunner.BatchCallableStatement		0%		0%	3	3	15	15	2	2	1	1
AsyncQueryRunner.UpdateCallableStatement		0%		0%	3	3	15	15	2	2	1	1
AsyncQueryRunner		89%		n/a	4	41	2	32	4	41	0	1
ResultSetIterator		70%		100%	1	10	8	21	1	8	0	1
StatementConfiguration		83%		93%	1	22	1	23	0	14	0	1
OutParameter		78%		100%	3	10	3	21	3	9	0	1
BasicRowProcessor		94%		66%	4	15	1	28	1	9	0	1
QueryLoader		95%		83%	1	9	1	22	0	6	0	1
DbUtils.DriverProxy		89%		n/a	1	8	1	10	1	8	0	1
MainClass		0%		n/a	1	1	2	2	1	1	1	1
StatementConfiguration.Builder		98%		50%	1	9	0	14	0	8	0	1
GenerousBeanProcessor		100%		91%	1	8	0	19	0	2	0	1
BasicRowProcessor.CaseInsensitiveHashMap		100%		n/a	0	6	0	15	0	6	0	1
ProxyFactory		100%		n/a	0	11	0	12	0	11	0	1
Total	1 683 of 4 292	60%	53 of 268	80%	264	562	457	1 129	219	428	4	20

Figure 5.5: Code coverage del package Dbutils post Randoop

Analizzando le classi che richiedevano miglieorie ai loro test, notiamo che:

- `AsyncQueryRunner` ha aumentato la instruction coverage del 2% e la branch coverage non poteva essere calcolata (n/a);
- `BaseResultSetHandler` è rimasta invariata sia per instruction che per branch coverage;
- `DbUtils` ha addirittura subito un decremento dell'8% per la instruction coverage e del 5% per la branch coverage;
- `ResultSetIterator` ha subito un incremento del 4% alla instruction coverage, mentre la branch coverage è rimasta invariata al 100%;
- `StatementConfiguration` ha guadagnato l'1% per la instruction coverage e il 6% sulla branch coverage.

Per le classi interne di `AsyncQueryRunner` non c'è stato modo di generare test automatici e di conseguenza la coverage non ha subito cambiamenti.

### 5.1.2 Randoop e PiTest

Analogalmente quanto fatto con `JaCoCo`, rigenerando il report di `PiTest` notiamo che le mutation coverage non è affatto cambiata.

## Pit Test Coverage Report

### Package Summary

org.apache.commons.dbutils

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
14	59% 690/1167	45% 280/628	82% 280/342

### Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">AbstractQueryRunner.java</a>	78% 135/174	74% 53/72	82% 53/65
<a href="#">AsyncQueryRunner.java</a>	33% 28/86	18% 12/67	38% 12/32
<a href="#">BaseResultSetHandler.java</a>	3% 8/292	2% 5/210	83% 5/6
<a href="#">BasicRowProcessor.java</a>	98% 42/43	81% 17/21	89% 17/19
<a href="#">BeanProcessor.java</a>	88% 111/126	87% 48/55	91% 48/53
<a href="#">DbUtils.java</a>	66% 65/98	46% 25/54	74% 25/34
<a href="#">GenerousBeanProcessor.java</a>	100% 19/19	100% 11/11	100% 11/11
<a href="#">MainClass.java</a>	0% 0/4	0% 0/1	100% 0/0
<a href="#">OutParameter.java</a>	86% 18/21	57% 4/7	100% 4/4
<a href="#">ProxyFactory.java</a>	100% 12/12	100% 9/9	100% 9/9
<a href="#">QueryLoader.java</a>	96% 22/23	75% 6/8	86% 6/7
<a href="#">QueryRunner.java</a>	87% 183/211	78% 58/74	84% 58/69
<a href="#">ResultSetIterator.java</a>	57% 12/21	45% 5/11	100% 5/5
<a href="#">StatementConfiguration.java</a>	95% 35/37	96% 27/28	96% 27/28

Figure 5.6: Mutation coverage del package Dbutils post Randoop

Il fatto che la mutation coverage non sia aumentata implica che i test generati non sono sufficientemente validi per catturare bug o comportamenti anomali.

### 5.1.3 Github Copilot e JaCoCo

Analogamente a quanto fatto con Randoop ho chiesto a Github-Copilot di migliorare le test suite per le stesse classi con cui Randoop ha lavorato (cioè quelle in `classes.txt`). Tuttavia, le cose non sono cambiate, infatti dopo aver buildato il progetto la coverage del package `dbutils` è rimasta la stessa rispetto a quella ottenuta con Randoop.

## 5. IL QUINTO TASK

### Apache Commons DbUtils

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.apache.commons.dbutils		60%		80%	264	562	457	1.129	219	428	4	20
org.apache.commons.dbutils.handlers		100%		100%	0	58	0	114	0	48	0	12
org.apache.commons.dbutils.wrappers		100%		93%	2	64	0	101	0	49	0	2
org.apache.commons.dbutils.handlers.columns		100%		82%	5	44	0	30	0	30	0	10
org.apache.commons.dbutils.handlers.properties		100%		94%	1	16	0	24	0	6	0	2
org.apache.commons.dbutils.constants		100%		n/a	0	1	0	2	0	1	0	1
Total	1.683 of 5.275	68%	61 of 364	83%	272	745	457	1.400	219	562	4	47

Figure 5.7: Code coverage del progetto post Github-Copilot

### org.apache.commons.dbutils

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
BaseResultSetHandler		2%		50%	188	193	283	291	187	192	0	1
QueryRunner		78%		82%	19	65	30	188	8	34	0	1
AbstractQueryRunner		82%		85%	14	62	36	174	5	32	0	1
DbUtils		55%		59%	7	31	25	82	2	20	0	1
AsyncQueryRunner.QueryCallableStatement		0%		0%	3	3	19	19	2	2	1	1
BeanProcessor		89%		85%	9	52	15	126	0	21	0	1
AsyncQueryRunner.BatchCallableStatement		0%		0%	3	3	15	15	2	2	1	1
AsyncQueryRunner.UpdateCallableStatement		0%		0%	3	3	15	15	2	2	1	1
AsyncQueryRunner		89%		n/a	4	41	2	32	4	41	0	1
ResultSetIterator		70%		100%	1	10	8	21	1	8	0	1
StatementConfiguration		83%		93%	1	22	1	23	0	14	0	1
OutParameter		78%		100%	3	10	3	21	3	9	0	1
BasicRowProcessor		94%		66%	4	15	1	28	1	9	0	1
QueryLoader		95%		83%	1	9	1	22	0	6	0	1
DbUtils.DriverProxy		89%		n/a	1	8	1	10	1	8	0	1
MainClass		0%		n/a	1	1	2	2	1	1	1	1
StatementConfiguration.Builder		98%		50%	1	9	0	14	0	8	0	1
GenerousBeanProcessor		100%		91%	1	8	0	19	0	2	0	1
BasicRowProcessor.CaseInsensitiveHashMap		100%		n/a	0	6	0	15	0	6	0	1
ProxyFactory		100%		n/a	0	11	0	12	0	11	0	1
Total	1.683 of 4.292	60%	53 of 268	80%	264	562	457	1.129	219	428	4	20

Figure 5.8: Code coverage del package dbutils post Github-Copilot

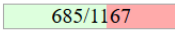
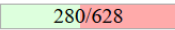
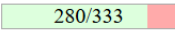
### 5.1.4 Github Copilot e PiTest

Per quanto riguarda PiTest, dopo l'utilizzo di Github-Copilot la mutation coverage del package dbutils è cambiata quanto segue:

## Pit Test Coverage Report

### Package Summary

org.apache.commons.dbutils

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
14	59% 	45% 	84% 

### Breakdown by Class

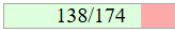
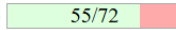
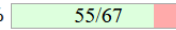
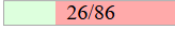
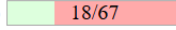
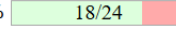
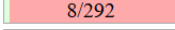
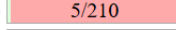
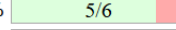
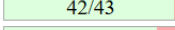
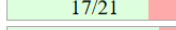
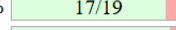
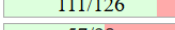
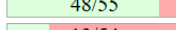
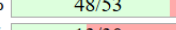
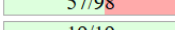
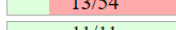
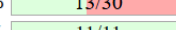
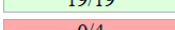
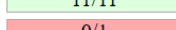
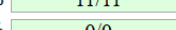
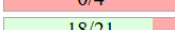
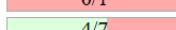
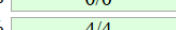
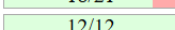
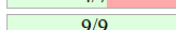
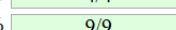
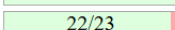
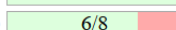
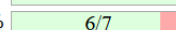
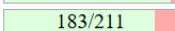
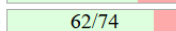
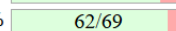
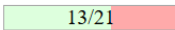
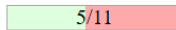
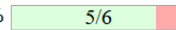
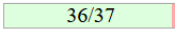
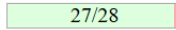
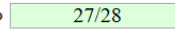



Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">AbstractQueryRunner.java</a>	79% 	76% 	82% 
<a href="#">AsyncQueryRunner.java</a>	30% 	27% 	75% 
<a href="#">BaseResultSetHandler.java</a>	3% 	2% 	83% 
<a href="#">BasicRowProcessor.java</a>	98% 	81% 	89% 
<a href="#">BeanProcessor.java</a>	88% 	87% 	91% 
<a href="#">DbUtils.java</a>	58% 	24% 	43% 
<a href="#">GenerousBeanProcessor.java</a>	100% 	100% 	100% 
<a href="#">MainClass.java</a>	0% 	0% 	100% 
<a href="#">OutParameter.java</a>	86% 	57% 	100% 
<a href="#">ProxyFactory.java</a>	100% 	100% 	100% 
<a href="#">QueryLoader.java</a>	96% 	75% 	86% 
<a href="#">QueryRunner.java</a>	87% 	84% 	90% 
<a href="#">ResultSetIterator.java</a>	62% 	45% 	83% 
<a href="#">StatementConfiguration.java</a>	97% 	96% 	96% 

Figure 5.9: Mutation coverage del package dbutils post Github-Copilot

- Per la classe `AsyncQueryRunner` la line coverage è scesa del 3%, la mutation coverage è aumentata del 9% e la test strength è salita di un sorprendente 37%;
- `BaseResultSetHandler` presenta gli stessi dati ottenuti con il primo report di PiTest, perciò nulla è cambiato;
- Per `DbUtils` c'è stato un forte decremento in quanto la line coverage è diminuita dell'8%, la mutation coverage del 22% e la test strength del 31%;
- `ResultSetIterator` presenta dei risultati misti siccome la line coverage è aumentata del 5%, la mutation coverage è rimasta invariata e la test strength è diminuita del 17%;
- Infine per `StatementConfiguration` la line coverage è aumentata del 2%, mentre la mutation coverage e la test strength non sono cambiate.

## 5. IL QUINTO TASK

---

Sommariamente nel package analizzato, line coverage e mutation coverage non sono cambiate, mentre la test strength è aumentata del 2%.



---

---

# CHAPTER 6

---

## IL SESTO TASK

### 6.1 Analisi della sicurezza e vulnerabilità del progetto

In questo capitolo si analizzeranno le problematiche in termini di sicurezza con `FindSecBugs` e vulnerabilità delle dipendenze usate tramite `Dependency-Check`.

#### 6.1.1 FindSecBugs

Dopo aver scaricato il tool, eseguendo il comando `./findsecbugs.sh percorsoProgetto/target/classes` è stato generato un report che segnale tutte le vulnerabilità riguardanti la sicurezza del progetto.

## 6. IL SESTO TASK

---

```
Lenovo@Andrea-V15 MINGW64 ~/Desktop/findsecbugs-cli-1.12.0
$ ./findsecbugs.sh C:/Users/Lenovo/Desktop/Magistrale/1-anno-I-semester/Software
-dependability/progetto_software_dependability/progettoSwD/progetto-software-dep
endability/target/classes
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#noProviders for further details.
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by edu.umd.cs.findbugs.ba.js
r305.TypeQualifierValue (file:/C:/Users/Lenovo/Desktop/findsecbugs-cli-1.12.0/li
b/spotbugs-4.6.0.jar)
WARNING: Please consider reporting this to the maintainers of edu.umd.cs.findbug
s.ba.jsr305.TypeQualifierValue
WARNING: System::setSecurityManager will be removed in a future release
M S ERRMSG: Possible information exposure through an error message At DbUtils.j
ava:[line 324]
M S SECSQLJDBC: This use of java/sql/Connection.prepareStatement(Ljava/lang/Str
ing;)Ljava/sql/CallableStatement; can be vulnerable to SQL injection (with JDBC) At
AbstractQueryRunner.java:[line 523]
M S SECSQLJDBC: This use of java/sql/Connection.prepareStatement(Ljava/lang/Str
ing;)Ljava/sql/PreparedStatement; can be vulnerable to SQL injection (with JDBC)
At AbstractQueryRunner.java:[line 607]
M S SECSQLJDBC: This use of java/sql/Statement.executeUpdate(Ljava/lang/String;
I)I can be vulnerable to SQL injection (with JDBC) At QueryRunner.java:[line 40
8]
M S SECSQLJDBC: This use of java/sql/Statement.executeUpdate(Ljava/lang/String;
)I can be vulnerable to SQL injection (with JDBC) At QueryRunner.java:[line 782
]
M S SECSQLJDBC: This use of java/sql/Connection.prepareStatement(Ljava/lang/Str
ing;)Ljava/sql/PreparedStatement; can be vulnerable to SQL injection (with JDBC)
At QueryRunner.java:[line 402]
```

Figure 6.1: Report FindSecBugs

Dalla figura di sopra notiamo che sono state trovate sei vulnerabilità. Analizziamole in maniera dettagliata:

1. **ERRMSG: Possible information exposure through an error message At DbUtils.java:[line 324]:** Questa issue riguarda la possibile esposizione di informazioni sensibili (ad esempio rilevare l'architettura del database) tramite messaggi di errore come si può vedere nel codice sottostante:

## 6. IL SESTO TASK

```
public static void printStackTrace(final SQLException e, final PrintWriter pw) {  
  
    SQLException next = e;  
    while (next != null) {  
        next.printStackTrace(pw);  
        next = next.getNextException();  
        if (next != null) {  
            pw.println("Next SQLException:");  
        }  
    }  
}
```

Figure 6.2: Potenziale vulnerabilità sugli errori

La soluzione che ho trovato è stata sostituire `printStackTrace` con una gestione dei log che scrive messaggi generici come "A database error occurred", evitando di esporre dettagli di eccezioni SQL;

```
public static void printStackTrace(final SQLException e, final PrintWriter pw) {  
    SQLException next = e;  
    while (next != null) {  
        pw.println("A database error occurred: " + next.getMessage()); // Messaggio generico  
        next = next.getNextException();  
        if (next != null) {  
            pw.println("Next SQLException:");  
        }  
    }  
}
```

Figure 6.3: Potenziale SQL Injection

2. **SECSQLIJDBC:** This use of `java/sql/Connection.prepareStatement` can be vulnerable to SQL injection (with JDBC) At `AbstractQueryRunner.java:[line 523]`: Questa issue segnala un potenziale pericolo di SQL Injection nel metodo `prepareCall`

```
protected CallableStatement prepareCall(final Connection conn, final String sql) 2 usages Gary Gregory  
    throws SQLException {  
  
    return conn.prepareStatement(sql);  
}
```

Figure 6.4: Potenziale SQL Injection

Per risolvere il problema è stata aggiunta la gestione parametrizzata

dei dati in `CallableStatement`, utilizzando il metodo `setObject` per passare in sicurezza i parametri alla query, evitando che input dinamici siano concatenati direttamente nella stringa SQL;

```
protected CallableStatement prepareCall(final Connection conn, final String sql, final Object... parameters)
    throws SQLException {
    CallableStatement stmt = conn.prepareCall(sql);
    for (int i = 0; i < parameters.length; i++) {
        stmt.setObject(parameterIndex: i + 1, parameters[i]); // Imposta i parametri per la query
    }
    return stmt;
}
```

Figure 6.5: Soluzione alla SQL Injection

3. **SECSQLIJDBC: This use of java/sql/Connection.prepareStatement can be vulnerable to SQL injection (with JDBC) At AbstractQueryRunner.java:[line 607]:** Anche questo è un caso che deriva da una possibile SQL Injection, però dipende dal formato della stringa contenente l'istruzione SQL che viene passata (se non contiene parametri allora non è possibile fare SQL Injection). Per risolvere il problema, è bastato parametrizzare tutte le query SQL centralizzando l'uso di `PreparedStatement` in un metodo che assegna automaticamente i parametri ai placeholder '?. In questo modo si evita la concatenazione diretta degli input nella query, prevenendo così il rischio di SQL Injection;
4. **SECSQLIJDBC: This use of java/sql/Statement.executeUpdate can be vulnerable to SQL injection (with JDBC) At QueryRunner.java:[line 408]:** Il problema segnalato riguarda l'uso di `Statement.executeUpdate(sql)` nel codice. Questo metodo è vulnerabile alla SQL Injection quando la query `sql` viene costruita dinamicamente e contiene input non sanitizzati, come ad esempio dati provenienti da input utente. Per risolvere questo problema, ho fatto in modo di utilizzare sempre un `PreparedStatement` anche nel caso in cui non ci siano parametri, invece di usare l'oggetto `Statement`. Il `PreparedStatement` è sicuro perché i parametri vengono gestiti

separatamente dalla query SQL, evitando la possibilità di iniettare codice SQL dannoso;

5. **SECSQLIJDBC: This use of java/sql/Statement.executeUpdate can be vulnerable to SQL injection (with JDBC) At QueryRunner.java:[line 782]:** Anche in questo caso ho provato a risolvere usando PreparedStatement anche quando non ci sono parametri (params == null OR params.length == 0) e viene utilizzato Statement.executeUpdate(sql) che può essere soggetto a SQL Injection;

```
Statement stmt = null;
int rows = 0;

try {
    if (params != null && params.length > 0) {
        final PreparedStatement ps = this.prepareStatement(conn, sql);
        stmt = ps;
        this.fillStatement(ps, params);
        rows = ps.executeUpdate();
    } else {
        stmt = conn.createStatement();
        rows = stmt.executeUpdate(sql);
    }
} catch (final SQLException e) {
    rethrow(e, sql, params);
}
```

Figure 6.6: Possibile SQL Injection quando non ci sono parametri

```

try {
    if (params != null && params.length > 0) {
        ps = this.prepareStatement(conn, sql);
        this.fillStatement(ps, params);
        rows = ps.executeUpdate();
    } else {
        ps = this.prepareStatement(conn, sql);
        rows = ps.executeUpdate();
    }
} catch (final SQLException e) {
    rethrow(e, sql, params);
}

```

Figure 6.7: Soluzione proposta

6. **SECSQLIJDBC**: This use of `java/sql/Connection.prepareStatement` can be vulnerable to SQL injection (with JDBC) At `QueryRunner.java:[line 402]`: Anche qui, ho deciso di utilizzare sempre un `PreparedStatement` per eseguire le query, sia quando ci sono parametri che quando non ce ne sono.

Dopo aver buildato il progetto e rilanciato il comando per l'analisi delle vulnerabilità notiamo che l'unica ad essere scomparsa è stata la numero 1. Per le SQL Injections, il fatto che siano ancora presenti anche dopo aver apportato le modifiche al codice, potrebbe essere un segnale di false positività da parte del tool.

```

$ ./findsecbugs.sh C:/Users/Lenovo/Desktop/FindSecBugs-CLI-1.12.0
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#noProviders for further details.
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by edu.umd.cs.findbugs.ba.jsr305.TypeQualifierValue (file:/C:/Users/Lenovo/Desktop/FindSecBugs-CLI-1.12.0/lib/spotbugs-4.6.0.jar)
WARNING: Please consider reporting this to the maintainers of edu.umd.cs.findbugs.ba.jsr305.TypeQualifierValue
WARNING: System::setSecurityManager will be removed in a future release
W S SECSQLIJDBC: This use of java/sql/Connection.prepareStatement(Ljava/lang/String;)Ljava/sql/CallableStatement; can be vulnerable to SQL injection (with JDBC) At AbstractQueryRunner.java:[line 522]
W S SECSQLIJDBC: This use of java/sql/Statement.executeUpdate(Ljava/lang/String;)I can be vulnerable to SQL injection (with JDBC) At QueryRunner.java:[line 408]
W S SECSQLIJDBC: This use of java/sql/Statement.executeUpdate(Ljava/lang/String;)I can be vulnerable to SQL injection (with JDBC) At QueryRunner.java:[line 392]
W S SECSQLIJDBC: This use of java/sql/Connection.prepareStatement(Ljava/lang/String;)Ljava/sql/PreparedStatement; can be vulnerable to SQL injection (with JDBC) At AbstractQueryRunner.java:[line 609]
W S SECSQLIJDBC: This use of java/sql/Connection.prepareStatement(Ljava/lang/String;)Ljava/sql/PreparedStatement; can be vulnerable to SQL injection (with JDBC) At QueryRunner.java:[line 402]

```

Figure 6.8: Report finale di FindSecsBugs

### 6.1.2 Dependency-Check

Tramite l'uso di questo tool sono riuscito a risalire alle dipendenze vulnerabili presenti nel mio progetto, identificando le librerie che presentavano problemi di sicurezza noti. Dopo aver scaricato Dependency-Check ho runnato il comando `./dependency-check.sh -s percorsoProgetto -f HTML` ed è stato generato un report in formato HTML.

**Project:**

Scan Information ([show all](#))

- `dependency-check` version: 8.2.1
- Report Generated On: Sat, 14 Dec 2024 18:27:58 +0100
- Dependencies Scanned: 48 (36 unique)
- Vulnerable Dependencies: 3
- Vulnerabilities Found: 3
- Vulnerabilities Suppressed: 0
- ...

**Summary**

Display: [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
<a href="#">covered-class:4.3.3-javadoc.jar.jszip.js</a>		<a href="#">pkg.javascript.jszip@3.7.1</a>	HIGH	1		3
<a href="#">covered-class:4.3.3-javadoc.jar.jszip.min.js</a>		<a href="#">pkg.javascript.jszip@3.7.1</a>	HIGH	1		3
<a href="#">replacecall:4.3.3.jar (shaded: org.apache.bcel/bcel@6.5.0)</a>	<a href="#">cpe:2.3:a:apache:commons_bcel:6.5.0:*:*:*:*:*</a>	<a href="#">pkg.maven/org.apache.bcel/bcel@6.5.0</a>	CRITICAL	1	Low	54

Figure 6.9: Report di Dependency-Check

Il report generato ha fornito dettagli utili, come il punteggio di gravità delle vulnerabilità (CVSS) e le versioni sicure delle dipendenze, permettendomi di intervenire per ridurre i rischi.

---

# CHAPTER 7

---

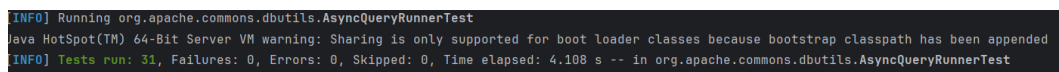
## IL SETTIMO TASK

### 7.1 Java Microbenchmark Harness

Nell'ultima parte di questo report analizzerò quelli che sono i componenti più "ingombranti" del progetto, ad esempio test suite che necessitano di troppo tempo per essere eseguite.

#### 7.1.1 Microbenchmarking su AsyncQueryRunner

Dopo aver fatto la build del progetto, analizzando i tempi richiesti dalle varie test suite, ho notato che quella di `AsyncQueryRunner` richiedeva molto più tempo rispetto a tutte le altre.



```
[INFO] Running org.apache.commons.dbutils.AsyncQueryRunnerTest
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
[INFO] Tests run: 31, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.108 s -- in org.apache.commons.dbutils.AsyncQueryRunnerTest
```

Figure 7.1

Con l'aiuto di Github-Copilot ho scritto delle classi di Benchmark che sono:

- **ArrayHandlerBenchmark**: misura quanto tempo ci vuole per eseguire



## 7. IL SETTIMO TASK

una query semplice (`SELECT 1`) su un database e per processare il `ResultSet` usando l'`ArrayHandler`;

- **AsyncQueryRunnerBenchmark**: misura il tempo medio necessario per eseguire una query su un database utilizzando `AsyncQueryRunner`. Sebbene la query venga eseguita in modo asincrono, il metodo `.get()` fa sì che il codice attenda il completamento dell'operazione prima di procedere, quindi il tempo misurato rappresenta il tempo totale per eseguire la query e ottenere il risultato, inclusa l'attesa per il completamento.;
- **QueryRunnerBenchmark**: misura il tempo medio necessario per eseguire una query su un database utilizzando la classe `QueryRunner`. A differenza del benchmark precedente con, qui l'esecuzione della query è sincrona. Il tempo misurato rappresenta quanto tempo impiega la query a essere eseguita e a restituire il risultato, con l'ausilio di un handler `ArrayHandler` che converte i risultati in un array.

Tuttavia, quando lancio il comando `mvn package` la build fallisce con il seguente errore:

```
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 30.799 s
[INFO] Finished at: 2024-12-23T19:29:18+01:00
[INFO] -----
[ERROR] Failed to execute goal org.moditect:moditect-maven-plugin:1.2.2.Final:add-module-info (add-module-infos) on project commons-dbutils: Execution add-module-infos of goal org.moditect:moditect-maven-plugin:1.2.2.Final:add-module-info failed: Invocation of jdeps failed: jdeps --generate-module-info C:\Users\Lenovo\Desktop\Magistrale\1-anno-I-semestre\Software-dependability\progetto_software_dependability\progetto-software-dependability\target\moditect --add-modules org.junit.platform.commons,jopt.simple,maven.model,jcommander,org.junit.platform.launcher,org.opentest4j,javax.annotation.api,pitest,junit5.plugin,aether.api,plexus.classworlds,aether.util,maven.plugin.api,org.junit.platform.engine,org.eclipse.sisu.inject,org.apache.commons.lang3 --module-path C:\Users\Lenovo\.m2\repository\org\junit\platform\junit-platform-commons\1.11.2\junit-platform-commons-1.11.2.jar;C:\Users\Lenovo\.m2\repository\net\sf\jopt-simple\jopt-simple\5.0.4\jopt-simple-5.0.4.jar;C:\Users\Lenovo\.m2\repository\org\apache\maven\maven-model\3.8.6\maven-model-3.8.6.jar;C:\Users\Lenovo\.m2\repository\org\jcommander\jcommander\1.83\jcommander-1.83.jar;C:\Users\Lenovo\.m2\repository\org\junit\platform\junit-platform-launcher\1.11.2\junit-platform-launcher-1.11.2.jar;C:\Users\Lenovo\.m2\repository\org\opentest4j\opentest4j\1.3.0\opentest4j-1.3.0.jar;C:\Users\Lenovo\.m2\repository\javax\annotation\javax.annotation-api\1.2\javax.annotation-api-1.2.jar;C:\Users\Lenovo\.m2\repository\org\pitest\pitest-junit5-plugin\1.2.1\pitest-junit5-plugin-1.2.1.jar;C:\Users\Lenovo\.m2\repository\org\eclipse\aether\aether-api\1.2.0\aether-api-1.2.0.jar;C:\Users\Lenovo\.m2\repository\org\codehaus\plexus\plexus-classworlds\2.6.0\plexus-classworlds-2.6.0.jar;C:\Users\Lenovo\.m2\repository\org\eclipse\aether\aether-util\1.0\aether-util-1.0.jar;C:\Users\Lenovo\.m2\repository\org\apache\maven\maven-plugin-api\3.8.6\maven-plugin-api-3.8.6.jar;C:\Users\Lenovo\.m2\repository\org\junit\platform\junit-platform-engine\1.11.2\junit-platform-engine-1.11.2.jar;C:\Users\Lenovo\.m2\repository\org\moditect\moditect\1.2.1\Final.jar;C:\Users\Lenovo\.m2\repository\org\openjdk\mh\jmh-core\1.37\jmh-core-1.37.jar;C:\Users\Lenovo\.m2\repository\org\codehaus\plexus\plexus-utils\3.3.1\plexus-utils-3.3.1.jar;C:\Users\Lenovo\.m2\repository\org\spiguardian\spiguardian-api\1.1.2\spiguardian-api-1.1.2.jar;C:\Users\Lenovo\.m2\repository\org\apache\commons\commons-math3\3.6.1\commons-math3-3.6.1.jar;C:\Users\Lenovo\.m2\repository\org\openjdk\mh\jmh-generator-annprocess\1.37\jmh-generator-annprocess-1.37.jar;C:\Users\Lenovo\.m2\repository\org\moditect\moditect-maven-plugin\1.2.1\Final\moditect-maven-plugin-1.2.1.Final.jar;C:\Users\Lenovo\.m2\repository\org\eclipse\sisu\org.eclipse.sisu.plexus\0.3.5\org.eclipse.sisu.plexus-0.3.5.jar;C:\Users\Lenovo\.m2\repository\org\apache\maven\maven-artifact\3.8.6\maven-artifact-3.8.6.jar;C:\Users\Lenovo\.m2\repository\org\codehaus\plexus\plexus-component-annotations\1.5.5\plexus-component-annotations-1.5.5.jar;C:\Users\Lenovo\.m2\repository\org\eclipse\sisu\org.eclipse.sisu.inject\0.3.5\org.eclipse.sisu.inject-0.3.5.jar;C:\Users\Lenovo\.m2\repository\org\apache\commons\commons-lang3\3.8.1\commons-lang3-3.8.1.jar --multi-release-9 C:\Users\Lenovo\Desktop\Magistrale\1-anno-I-semestre\Software-dependability\progetto_software_dependability\progettoSwD\progetto-software-dependability\target\commons-dbutils-1.9.0-SNAPSHOT.jar -> [Help 1]
```

Figure 7.2

L'errore riguarda l'esecuzione del plugin `moditect-maven-plugin` che tenta di generare un file `module-info.java` per supportare il sistema di moduli Java, ma fallisce durante l'esecuzione.