Project Report: VroomRentals Mutation Testing Analysis

Contributors

- Shashank Siddappa Devarmani (IMT2022107)
- Aditya Saraf (IMT2022067)
- **Repository Link:** https://github.com/NikaYz/Car-Rental-System

0. Project Overview

VroomRentals is a car rental management system implemented in Java. The system utilizes a Service Layer architecture (Customer, Driver, Car, Rental, and User services) interacting with a MySQL database via a custom DatabaseManager.

Objective: To evaluate the robustness of the test suite not just by code coverage, but by its ability to detect faults (mutations). As noted in the reference material, traditional test coverage (line, branch) only measures which code is executed. It does not verify that the tests can actually detect faults within that executed code. Therefore, this project employs **Mutation Testing** to ensure high test adequacy.

1. Project Requirements & Execution Guide

To successfully build and run the VroomRentals mutation testing suite, the following environment configurations are mandatory.

1.1 System Requirements

- **Java Development Kit (JDK): Version 17** (Strict Requirement).
    - *Note:* Although the project source compatibility is set to 11 in the configuration, the execution environment and the latest Pitest/Mockito plugins function optimally with **Java 17**. Lower versions may result in plugin compatibility errors.
- **Build Tool: Apache Maven** (Version 3.8 or higher).
- **Integrated Development Environment (IDE):** IntelliJ IDEA or Eclipse (Optional, but recommended for viewing HTML reports).
- To run the code locally after testing, we would need to have mysql connector and keep running at port 3306, if not change it in DatabaseManager.java :

```
private static final String DB_URL = "jdbc:mysql://localhost:3306/vroomrentals?useSSL=false&allowPublicKeyRetrieval=true";
```

  And change the root and password based on your system requirements in the same class.

1.2 Project Dependencies (Managed via Maven)

The project relies on the following key libraries, which are automatically fetched by Maven:

- **JUnit 5 (Jupiter):** For defining and running the test suite.
- **Mockito 5.8.0:** For mocking the DatabaseManager in unit tests.
- **H2 Database Engine:** For running in-memory integration tests.
- **Pitest 1.15.0:** The core engine for generating mutants and calculating mutation scores.
- **MySQL Connector:** Included for potential production connectivity (though tests use H2).

1.3 How to Run the Project (be at main folder)

Step 1: Verify Installation Ensure Java 17 and Maven are correctly installed:

```
java -version (Must show version 17)

mvn -version
```

Step 2: Clean and Build Navigate to the project root directory containing the pom.xml file and run: mvn clean install

Step 3: Run Standard Unit & Integration Tests To execute the JUnit test suite without mutation coverage:

```
mvn test
```

Step 4: Execute Mutation Testing To trigger Pitest, generate mutants, and create the coverage report:

```
mvn org.pitest:pitest-maven:mutationCoverage
```

Step 5: View Reports After the mutation run completes, the HTML reports will be generated in the target directory. Open the following file in a browser to view the results: target/pit-reports/index.html

2. Methodology & Tooling

2.1 Mutation Tool: Pitest

In accordance with the project requirements, we utilized **Pitest** for Java mutation testing. While the reference material discusses tools like Stryker (JS) and MutPy (Python), Pitest provides equivalent functionality for Java, automatically seeding faults into the bytecode and running JUnit 5 tests against them.

2.2 Testing Frameworks: Mockito & H2

To effectively implement the "Unit" and "Integration" testing levels defined in the project guidelines, we selected specific frameworks to handle dependencies:

Why Mockito? (Unit Testing)

For Unit Mutation Testing, the goal is to test individual components (e.g., CarService) in isolation.

- **Isolation:** We used Mockito to create mock objects of the DatabaseManager. This ensures that our unit tests focus solely on the business logic of the service layer (e.g., "Did the service construct the correct SQL string?") without requiring a running database.
- **Determinism:** By mocking the database response (e.g., defining exactly what rs.next() returns), we eliminate flaky tests caused by external factors.
- **Implementation:** In CarServiceTest.java, strict argument matchers (e.g., argThat) verify that the service generates precise SQL queries, allowing us to detect logic errors immediately.

Why H2 Database? (Integration Testing)

For Integration Mutation Testing, the goal is to verify the connections between components—specifically between the Service Layer and the Database Layer.

- **Realistic Simulation:** We used H2 in an in-memory mode (jdbc:h2:mem:testdb;MODE=MySQL) to simulate the production MySQL environment. This allows us to test actual SQL execution, constraints, and state changes (INSERT/UPDATE) without the overhead of configuring a live MySQL server.
- **Speed & Ephemerality:** H2 runs entirely in memory, making the integration tests fast enough to run repeatedly during mutation analysis. The database is created and destroyed with each test run, ensuring a clean state for every test case.
- **Implementation:** IntegrationTest.java utilizes H2 to verify that valid SQL commands are not only generated but successfully executed against a compliant schema.

3. Mutation Operators Applied

To meet the requirement of using at least three different operators at both Unit and Integration levels, the following operators were targeted.

3.1 Unit Level Operators

These operators focus on individual components and internal logic.

- **Math Mutator (Arithmetic Operator Replacement):**
  - *Concept:* Replaces arithmetic operations (e.g., + with *, - with /).
  - *Application:* In DriverService.java, the calculateScore method was specifically designed to test this operator. Pitest replaces currentScore++ with currentScore-- to test if the suite detects the math error.
- **Conditionals Boundary Mutator (Relational Operator Replacement):**
  - *Concept:* Replaces boolean relations, such as changing > to >= or == to <=.
  - *Application:* RentalService.java relies heavily on logic like YEAR(rental_start) = ... Mutation tests altered these boundaries to ensure our tests (e.g., getRentalCount_shouldReturnCorrectCount) strictly enforced date boundaries.
- **Return Values Mutator (Statement Deletion/Modification):**

    - *Concept:* Related to "Remove method body" or changing return values to defaults (null/0).
    - *Application:* In RentalServiceTest.java, we observed surviving mutants where the method returned null instead of a ResultSet. We explicitly added assertNotNull(result) to kill these specific mutants (See Section 5).

3.2 Integration Level Operators

Integration mutation targets the connections between components. In VroomRentals, this occurs at the interface between the Service Layer and the Database Layer.

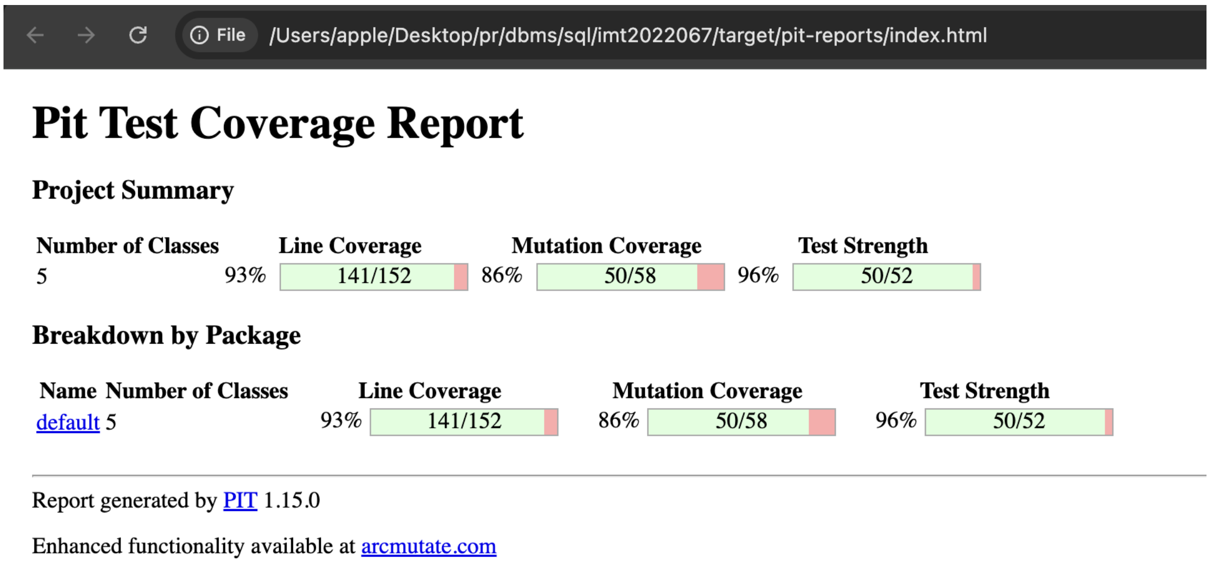
- **Non-Void Method Calls (Integration Method Call Deletion - IMCD):**
    - *Concept:* Deleting calls to methods that return values.
    - *Application:* In IntegrationTest.java, we test the driverService.addDriver method. If the call to dbManager.executeUpdate() is removed (mutated), the database state will not change. Our integration tests verify the actual H2 database state to kill this mutant.
- **Argument Propagation (Integration Parameter Exchange - IPEX):**
    - *Concept:* Swapping parameters in method calls.
    - *Application:* In CustomerService.addCustomer, parameters fname and lname are passed to the SQL query. If these are swapped, the data integrity is lost. The IntegrationTest verifies that "Alice" (first name) is not stored as the last name.
- **SQL String Mutation (Specific to JDBC):**
    - *Concept:* Modifying the data flow passed to external systems.
    - *Application:* Since our DatabaseManager uses string concatenation, a mutation that changes the SQL string construction (e.g., dropping the WHERE clause) is treated as an integration fault. Our tests in UserServiceTest verify the exact SQL syntax using verify(mockDbManager).executeQuery(argThat(...)).

4. Results Summary

4.1 Overall Statistics

Based on the generated Pitest report, the project achieved the following aggregate scores:

- **Line Coverage:** 93% (141/152 lines)
- **Mutation Coverage:** 86% (50/58 mutants killed)
- **Test Strength:** 96%



/Users/apple/Desktop/pr/dbms/sql/imt2022067/target/pit-reports/index.html

Pit Test Coverage Report

Project Summary

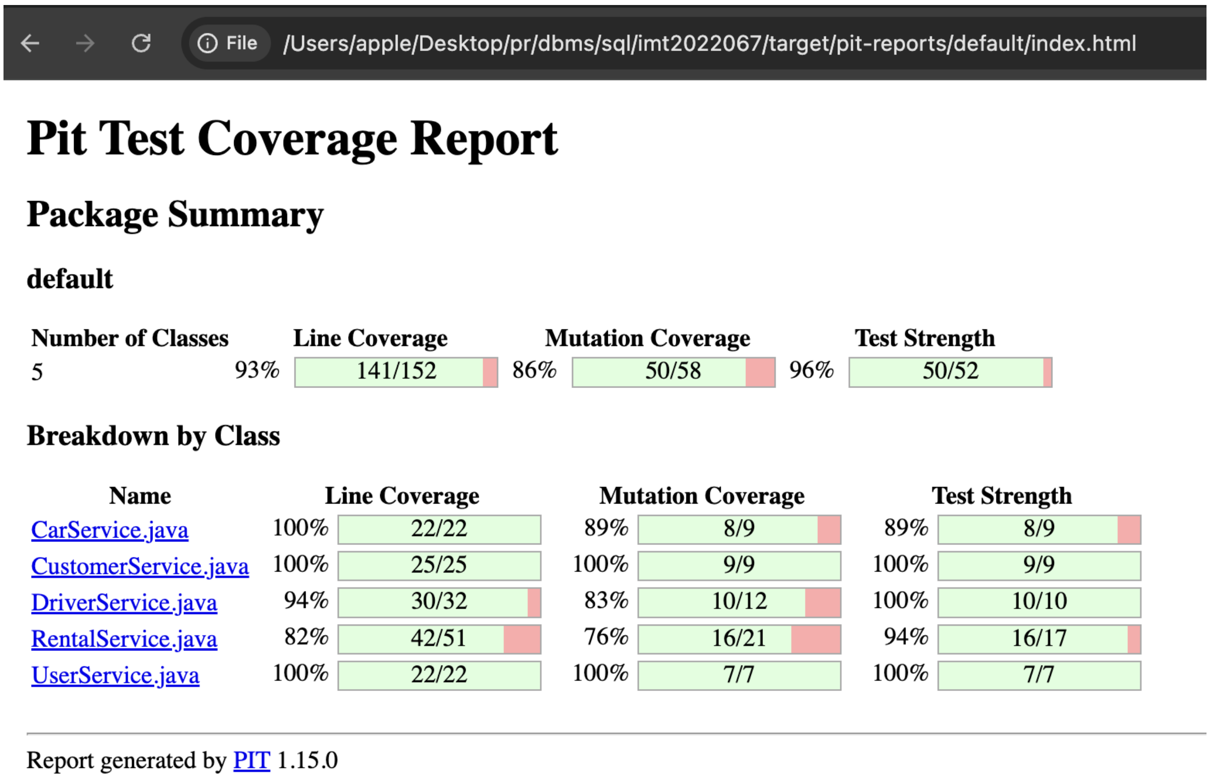
Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
5	93%	141/152	86%	50/58	96%	50/52

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
default	5	93%	141/152	86%	50/58	96%	50/52

Report generated by PIT 1.15.0

Enhanced functionality available at arcmutate.com



/Users/apple/Desktop/pr/dbms/sql/imt2022067/target/pit-reports/default/index.html

Pit Test Coverage Report

Package Summary

default

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
5	93%	141/152	86%	50/58	96%	50/52

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
CarService.java	100%	22/22	89%	8/9	89%	8/9
CustomerService.java	100%	25/25	100%	9/9	100%	9/9
DriverService.java	94%	30/32	83%	10/12	100%	10/10
RentalService.java	82%	42/51	76%	16/21	94%	16/17
UserService.java	100%	22/22	100%	7/7	100%	7/7

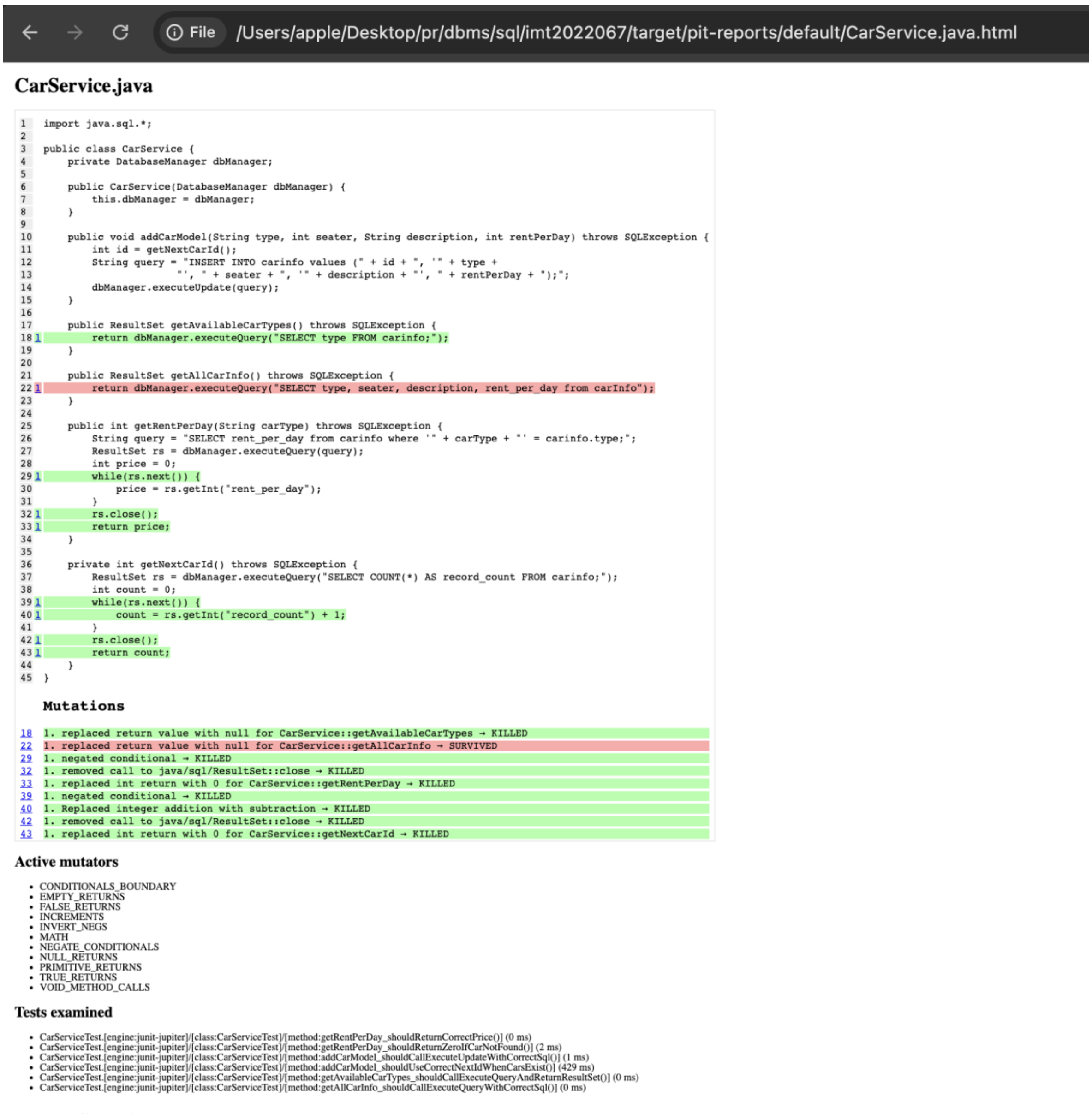
Report generated by PIT 1.15.0

This score indicates a highly robust test suite. As noted in the reference, the quality of tests can be gauged from the percentage of mutations killed, and experimental studies show this is a highly effective method for evaluating test adequacy.

4.2 Detailed Class Breakdown

The following screenshots detail the mutation coverage for each specific service class within the system.

CarService:



File /Users/apple/Desktop/pr/dbms/sql/imt2022067/target/pit-reports/default/CarService.java.html

CarService.java

```
1    import java.sql.*;
2
3    public class CarService {
4        private DatabaseManager dbManager;
5
6        public CarService(DatabaseManager dbManager) {
7            this.dbManager = dbManager;
8        }
9
10       public void addCarModel(String type, int seater, String description, int rentPerDay) throws SQLException {
11           int id = getNextCarId();
12           String query = "INSERT INTO carinfo values (" + id + ", '" + type +
13                          "', " + seater + ", '" + description + "', " + rentPerDay + ");";
14           dbManager.executeUpdate(query);
15       }
16
17       public ResultSet getAvailableCarTypes() throws SQLException {
18 1         return dbManager.executeQuery("SELECT type FROM carinfo;");
19       }
20
21       public ResultSet getAllCarInfo() throws SQLException {
22 1         return dbManager.executeQuery("SELECT type, seater, description, rent_per_day from carInfo");
23       }
24
25       public int getRentPerDay(String carType) throws SQLException {
26           String query = "SELECT rent_per_day from carinfo where '" + carType + "' = carinfo.type;";
27           ResultSet rs = dbManager.executeQuery(query);
28           int price = 0;
29 1         while(rs.next()) {
30               price = rs.getInt("rent_per_day");
31           }
32 1         rs.close();
33 1         return price;
34       }
35
36       private int getNextCarId() throws SQLException {
37           ResultSet rs = dbManager.executeQuery("SELECT COUNT(*) AS record_count FROM carinfo;");
38           int count = 0;
39 1         while(rs.next()) {
40 1             count = rs.getInt("record_count") + 1;
41           }
42 1         rs.close();
43 1         return count;
44       }
45   }
```

Mutations

```
18  1. replaced return value with null for CarService::getAvailableCarTypes → KILLED
22  1. replaced return value with null for CarService::getAllCarInfo → SURVIVED
29  1. negated conditional → KILLED
32  1. removed call to java/sql/ResultSet::close → KILLED
33  1. replaced int return with 0 for CarService::getRentPerDay → KILLED
39  1. negated conditional → KILLED
40  1. Replaced integer addition with subtraction → KILLED
42  1. removed call to java/sql/ResultSet::close → KILLED
43  1. replaced int return with 0 for CarService::getNextCarId → KILLED
```

Active mutators

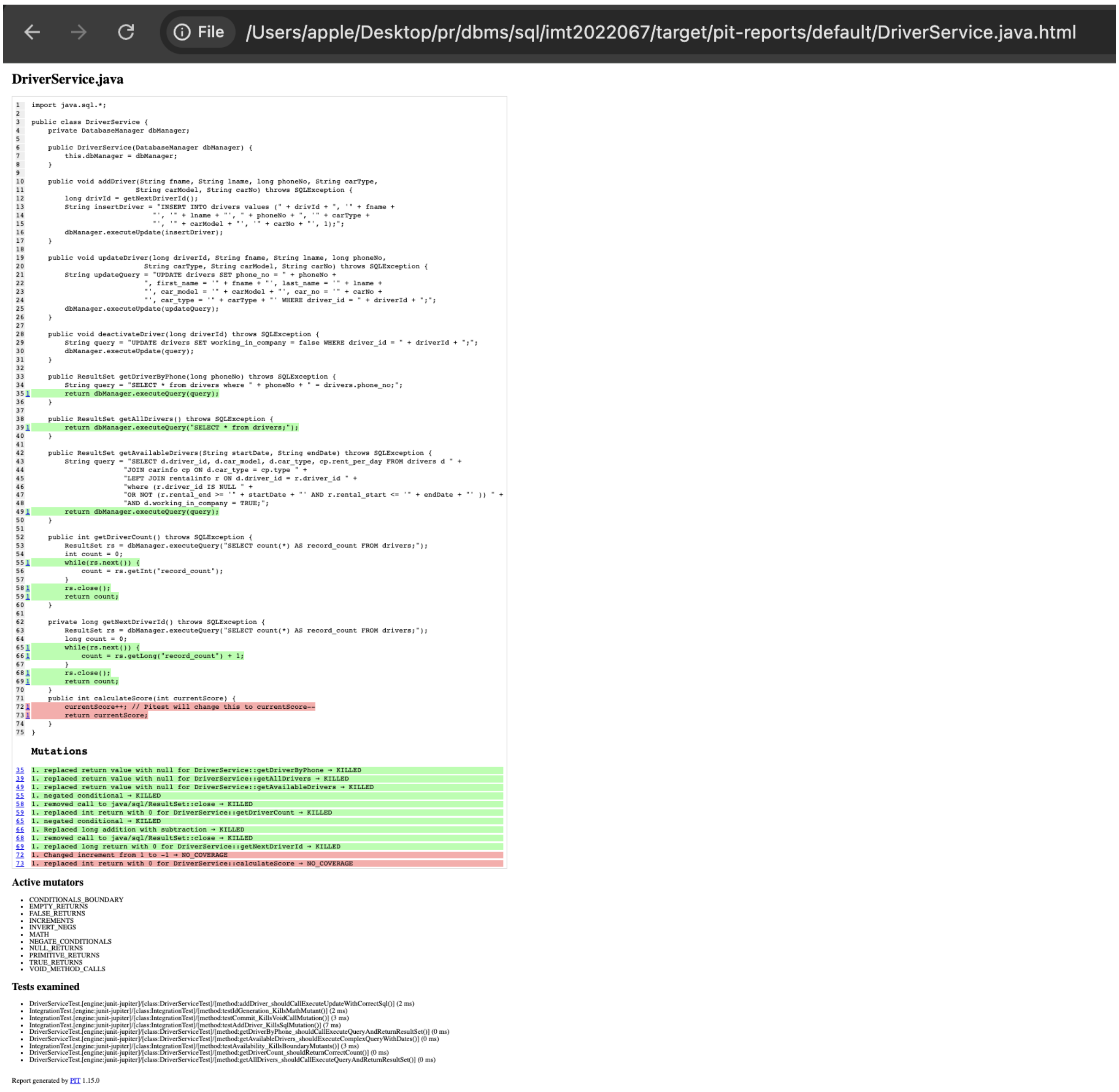
- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- CarServiceTest.[engine:junit-jupiter]/[class:CarServiceTest]/[method:getRentPerDay_shouldReturnCorrectPrice()] (0 ms)
- CarServiceTest.[engine:junit-jupiter]/[class:CarServiceTest]/[method:getRentPerDay_shouldReturnZeroIfCarNotFound()] (2 ms)
- CarServiceTest.[engine:junit-jupiter]/[class:CarServiceTest]/[method:addCarModel_shouldCallExecuteUpdateWithCorrectSql()] (1 ms)
- CarServiceTest.[engine:junit-jupiter]/[class:CarServiceTest]/[method:addCarModel_shouldUseCorrectNextIdWhenCarsExist()] (429 ms)
- CarServiceTest.[engine:junit-jupiter]/[class:CarServiceTest]/[method:getAvailableCarTypes_shouldCallExecuteQueryAndReturnResultSet()] (0 ms)
- CarServiceTest.[engine:junit-jupiter]/[class:CarServiceTest]/[method:getAllCarInfo_shouldCallExecuteQueryWithCorrectSql()] (0 ms)

- *Observation:* High coverage was achieved by mocking the DatabaseManager and verifying SQL string generation.

DriverService


File
/Users/apple/Desktop/pr/dbms/sql/imt2022067/target/pit-reports/default/DriverService.java.html

DriverService.java

```
1    import java.sql.*;
2
3    public class DriverService {
4        private DatabaseManager dbManager;
5
6        public DriverService(DatabaseManager dbManager) {
7            this.dbManager = dbManager;
8        }
9
10       public void addDriver(String fname, String lname, long phoneNo, String carType,
11                             String carModel, String carNo) throws SQLException {
12           long drivId = getNextDriverId();
13           String insertDriver = "INSERT INTO drivers values (" + drivId + ", '" + fname +
14                                 "', '" + lname + "', " + phoneNo + ", '" + carType +
15                                 "', '" + carModel + "', '" + carNo + "', 1);";
16           dbManager.executeUpdate(insertDriver);
17       }
18
19       public void updateDriver(long driverId, String fname, String lname, long phoneNo,
20                              String carType, String carModel, String carNo) throws SQLException {
21           String updateQuery = "UPDATE drivers SET phone_no = " + phoneNo +
22                              ", first_name = '" + fname + "', last_name = '" + lname +
23                              "', car_model = '" + carModel + "', car_no = '" + carNo +
24                              "', car_type = '" + carType + "' WHERE driver_id = " + driverId + ";";
25           dbManager.executeUpdate(updateQuery);
26       }
27
28       public void deactivateDriver(long driverId) throws SQLException {
29           String query = "UPDATE drivers SET working_in_company = false WHERE driver_id = " + driverId + ";";
30           dbManager.executeUpdate(query);
31       }
32
33       public ResultSet getDriverByPhone(long phoneNo) throws SQLException {
34           String query = "SELECT * from drivers where " + phoneNo + " = drivers.phone_no;";
35 1         return dbManager.executeQuery(query);
36       }
37
38       public ResultSet getAllDrivers() throws SQLException {
39 1         return dbManager.executeQuery("SELECT * from drivers;");
40       }
41
42       public ResultSet getAvailableDrivers(String startDate, String endDate) throws SQLException {
43           String query = "SELECT d.driver_id, d.car_model, d.car_type, cp.rent_per_day FROM drivers d " +
44                          "JOIN carinfo cp ON d.car_type = cp.type " +
45                          "LEFT JOIN rentalinfo r ON d.driver_id = r.driver_id " +
46                          "where (r.driver_id IS NULL " +
47                          "OR NOT (r.rental_end >= '" + startDate + "' AND r.rental_start <= '" + endDate + "' )) " +
48                          "AND d.working_in_company = TRUE;";
49 1         return dbManager.executeQuery(query);
50       }
51
52       public int getDriverCount() throws SQLException {
53           ResultSet rs = dbManager.executeQuery("SELECT count(*) AS record_count FROM drivers;");
54           int count = 0;
55 1         while(rs.next()) {
56               count = rs.getInt("record_count");
57           }
58 1         rs.close();
59 1         return count;
60       }
61
62       private long getNextDriverId() throws SQLException {
63           ResultSet rs = dbManager.executeQuery("SELECT count(*) AS record_count FROM drivers;");
64           long count = 0;
65 1         while(rs.next()) {
66 1             count = rs.getLong("record_count") + 1;
67           }
68 1         rs.close();
69 1         return count;
70       }
71       public int calculateScore(int currentScore) {
72 1         currentScore++; // Pitest will change this to currentScore--
73 1         return currentScore;
74       }
75   }
```

Mutations

```
35  1. replaced return value with null for DriverService::getDriverByPhone → KILLED
39  1. replaced return value with null for DriverService::getAllDrivers → KILLED
49  1. replaced return value with null for DriverService::getAvailableDrivers → KILLED
55  1. negated conditional → KILLED
58  1. removed call to java/sql/ResultSet::close → KILLED
59  1. replaced int return with 0 for DriverService::getDriverCount → KILLED
65  1. negated conditional → KILLED
66  1. Replaced long addition with subtraction → KILLED
68  1. removed call to java/sql/ResultSet::close → KILLED
69  1. replaced long return with 0 for DriverService::getNextDriverId → KILLED
72  1. Changed increment from 1 to -1 → NO_COVERAGE
73  1. replaced int return with 0 for DriverService::calculateScore → NO_COVERAGE
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

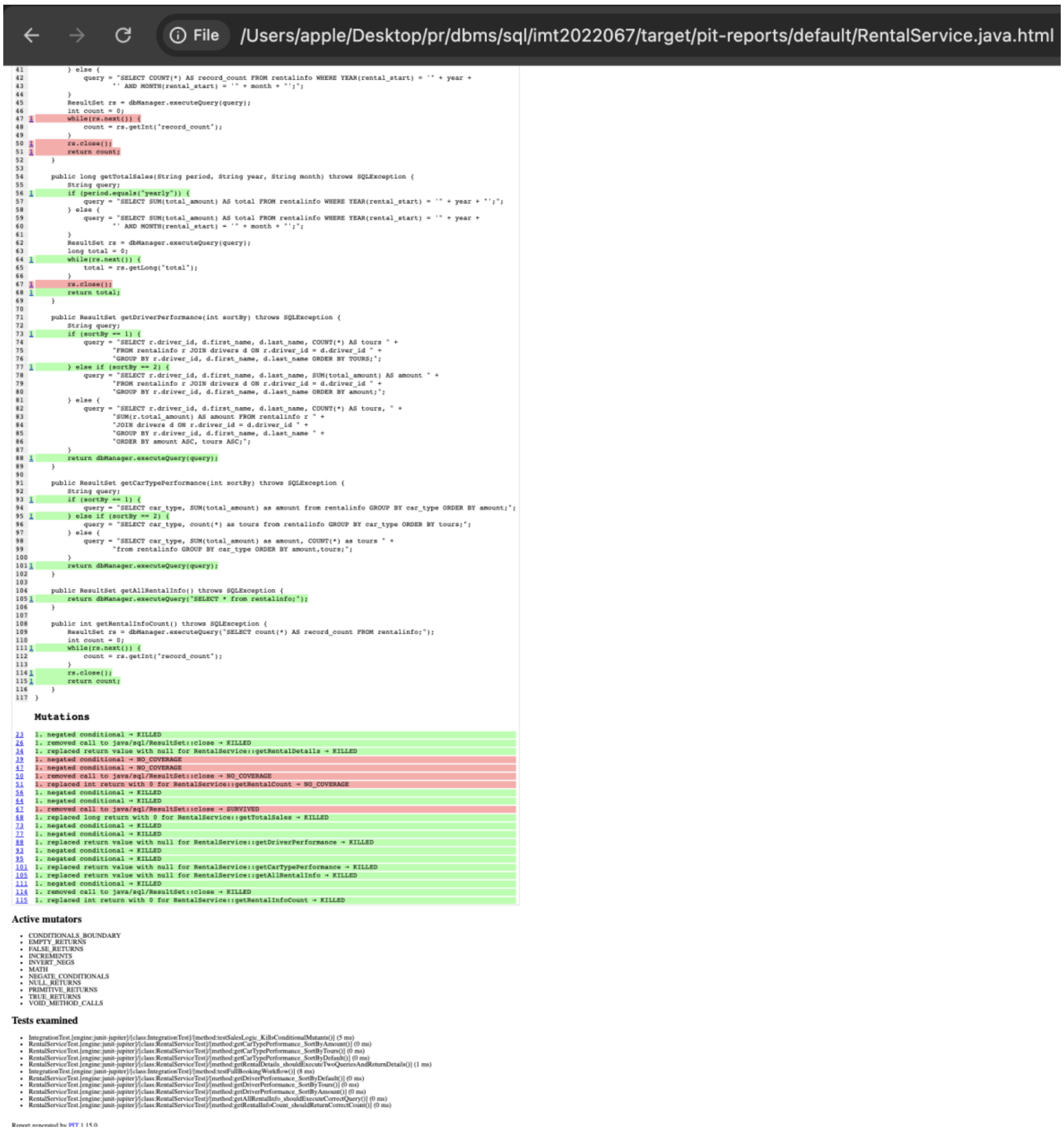
Tests examined

- DriverServiceTest.[engine:junit-jupiter]/[class:DriverServiceTest]/[method:addDriver_shouldCallExecuteUpdateWithCorrectSql()] (2 ms)
- IntegrationTest.[engine:junit-jupiter]/[class:IntegrationTest]/[method:testIdGeneration_KillsMathMutant()] (2 ms)
- IntegrationTest.[engine:junit-jupiter]/[class:IntegrationTest]/[method:testCommit_KillsVoidCallMutation()] (3 ms)
- IntegrationTest.[engine:junit-jupiter]/[class:IntegrationTest]/[method:testAddDriver_KillsSqlMutation()] (7 ms)
- DriverServiceTest.[engine:junit-jupiter]/[class:DriverServiceTest]/[method:getDriverByPhone_shouldCallExecuteQueryAndReturnResultSet()] (0 ms)
- DriverServiceTest.[engine:junit-jupiter]/[class:DriverServiceTest]/[method:getAvailableDrivers_shouldExecuteComplexQueryWithDates()] (0 ms)
- IntegrationTest.[engine:junit-jupiter]/[class:IntegrationTest]/[method:testAvailability_KillsBoundaryMutants()] (3 ms)
- DriverServiceTest.[engine:junit-jupiter]/[class:DriverServiceTest]/[method:getDriverCount_shouldReturnCorrectCount()] (0 ms)
- DriverServiceTest.[engine:junit-jupiter]/[class:DriverServiceTest]/[method:getAllDrivers_shouldCallExecuteQueryAndReturnResultSet()] (0 ms)

Report generated by PIT 1.15.0

- *Observation:* Includes the "Math Mutator" test where calculateScore logic was verified.

RentalService :


/Users/apple/Desktop/pr/dbms/sql/imt2022067/target/pit-reports/default/RentalService.java.html

```
41          } else {
42              query = "SELECT COUNT(*) AS record_count FROM rentalinfo WHERE YEAR(rental_start) = '" + year +
43                      "' AND MONTH(rental_start) = '" + month + "';";
44          }
45          ResultSet rs = dbManager.executeQuery(query);
46          int count = 0;
47  1       while(rs.next()) {
48              count = rs.getInt("record_count");
49          }
50  1       rs.close();
51  1       return count;
52      }
53
54      public long getTotalSales(String period, String year, String month) throws SQLException {
55          String query;
56  1       if (period.equals("yearly")) {
57              query = "SELECT SUM(total_amount) AS total FROM rentalinfo WHERE YEAR(rental_start) = '" + year + "';";
58          } else {
59              query = "SELECT SUM(total_amount) AS total FROM rentalinfo WHERE YEAR(rental_start) = '" + year +
60                      "' AND MONTH(rental_start) = '" + month + "';";
61          }
62          ResultSet rs = dbManager.executeQuery(query);
63          long total = 0;
64  1       while(rs.next()) {
65              total = rs.getLong("total");
66          }
67  1       rs.close();
68  1       return total;
69      }
70
71      public ResultSet getDriverPerformance(int sortBy) throws SQLException {
72          String query;
73  1       if (sortBy == 1) {
74              query = "SELECT r.driver_id, d.first_name, d.last_name, COUNT(*) AS tours " +
75                      "FROM rentalinfo r JOIN drivers d ON r.driver_id = d.driver_id " +
76                      "GROUP BY r.driver_id, d.first_name, d.last_name ORDER BY TOURS;";
77  1       } else if (sortBy == 2) {
78              query = "SELECT r.driver_id, d.first_name, d.last_name, SUM(total_amount) AS amount " +
79                      "FROM rentalinfo r JOIN drivers d ON r.driver_id = d.driver_id " +
80                      "GROUP BY r.driver_id, d.first_name, d.last_name ORDER BY amount;";
81          } else {
82              query = "SELECT r.driver_id, d.first_name, d.last_name, COUNT(*) AS tours, " +
83                      "SUM(r.total_amount) AS amount FROM rentalinfo r " +
84                      "JOIN drivers d ON r.driver_id = d.driver_id " +
85                      "GROUP BY r.driver_id, d.first_name, d.last_name " +
86                      "ORDER BY amount ASC, tours ASC;";
87          }
88  1       return dbManager.executeQuery(query);
89      }
90
91      public ResultSet getCarTypePerformance(int sortBy) throws SQLException {
92          String query;
93  1       if (sortBy == 1) {
94              query = "SELECT car_type, SUM(total_amount) as amount from rentalinfo GROUP BY car_type ORDER BY amount;";
95  1       } else if (sortBy == 2) {
96              query = "SELECT car_type, count(*) as tours from rentalinfo GROUP BY car_type ORDER BY tours;";
97          } else {
98              query = "SELECT car_type, SUM(total_amount) as amount, COUNT(*) as tours " +
99                      "from rentalinfo GROUP BY car_type ORDER BY amount,tours;";
100         }
101 1       return dbManager.executeQuery(query);
102     }
103
104     public ResultSet getAllRentalInfo() throws SQLException {
105 1       return dbManager.executeQuery("SELECT * from rentalinfo;");
106     }
107
108     public int getRentalInfoCount() throws SQLException {
109         ResultSet rs = dbManager.executeQuery("SELECT count(*) AS record_count FROM rentalinfo;");
110         int count = 0;
111 1       while(rs.next()) {
112             count = rs.getInt("record_count");
113         }
114 1       rs.close();
115 1       return count;
116     }
117 }
```

Mutations

```
23   1. negated conditional → KILLED
26   1. removed call to java/sql/ResultSet::close → KILLED
34   1. replaced return value with null for RentalService::getRentalDetails → KILLED
39   1. negated conditional → NO_COVERAGE
47   1. negated conditional → NO_COVERAGE
50   1. removed call to java/sql/ResultSet::close → NO_COVERAGE
51   1. replaced int return with 0 for RentalService::getRentalCount → NO_COVERAGE
56   1. negated conditional → KILLED
64   1. negated conditional → KILLED
67   1. removed call to java/sql/ResultSet::close → SURVIVED
68   1. replaced long return with 0 for RentalService::getTotalSales → KILLED
73   1. negated conditional → KILLED
77   1. negated conditional → KILLED
88   1. replaced return value with null for RentalService::getDriverPerformance → KILLED
93   1. negated conditional → KILLED
95   1. negated conditional → KILLED
101  1. replaced return value with null for RentalService::getCarTypePerformance → KILLED
105  1. replaced return value with null for RentalService::getAllRentalInfo → KILLED
111  1. negated conditional → KILLED
114  1. removed call to java/sql/ResultSet::close → KILLED
115  1. replaced int return with 0 for RentalService::getRentalInfoCount → KILLED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

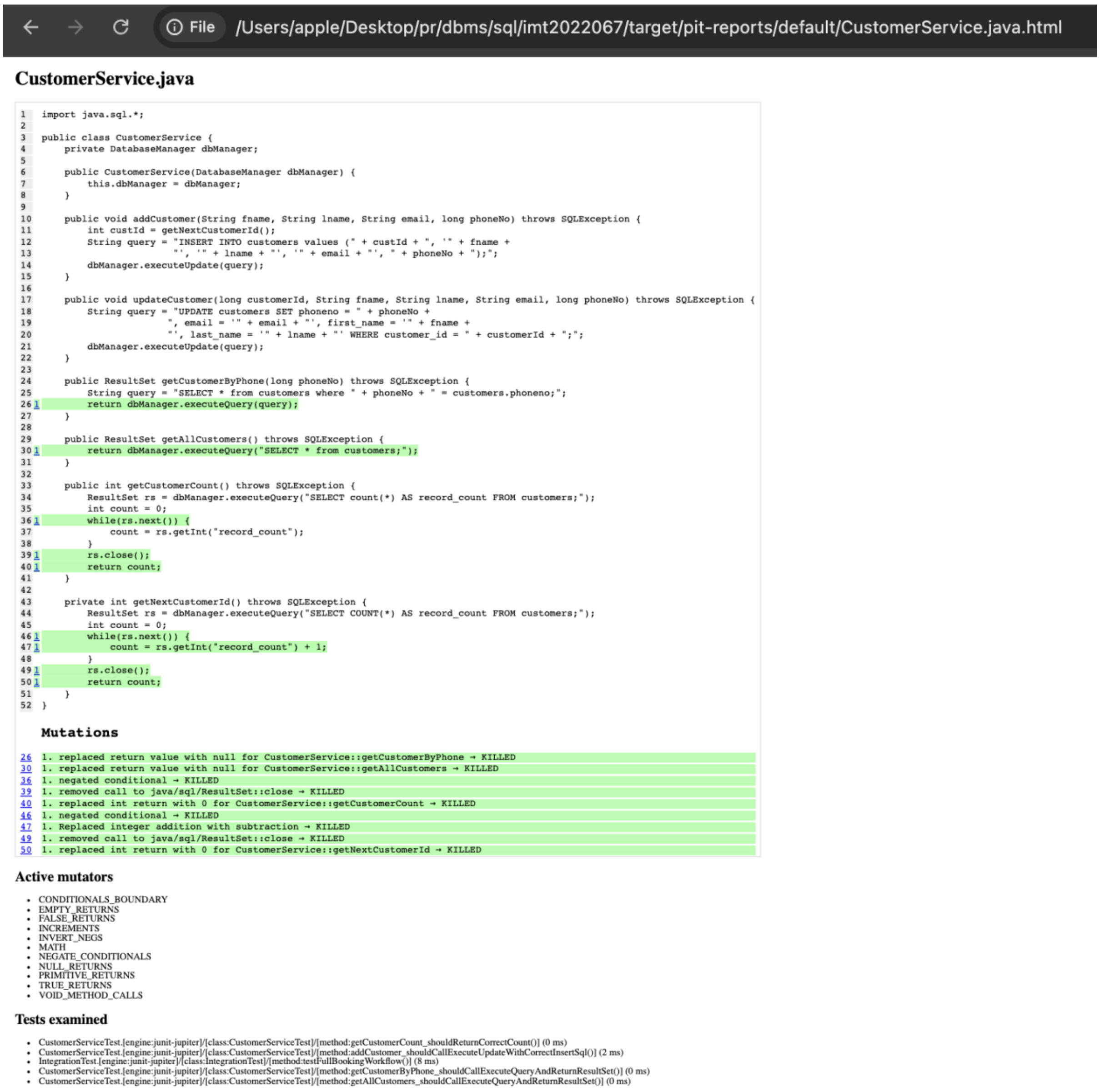
Tests examined

- IntegrationTest.[engine:junit-jupiter]/[class:IntegrationTest]/[method:testSalesLogic_KillsConditionalMutants()] (5 ms)
- RentalServiceTest.[engine:junit-jupiter]/[class:RentalServiceTest]/[method:getCarTypePerformance_SortByAmount()] (0 ms)
- RentalServiceTest.[engine:junit-jupiter]/[class:RentalServiceTest]/[method:getCarTypePerformance_SortByTours()] (0 ms)
- RentalServiceTest.[engine:junit-jupiter]/[class:RentalServiceTest]/[method:getCarTypePerformance_SortByDefault()] (0 ms)
- RentalServiceTest.[engine:junit-jupiter]/[class:RentalServiceTest]/[method:getRentalDetails_shouldExecuteTwoQueriesAndReturnDetails()] (1 ms)
- IntegrationTest.[engine:junit-jupiter]/[class:IntegrationTest]/[method:testFullBookingWorkflow()] (8 ms)
- RentalServiceTest.[engine:junit-jupiter]/[class:RentalServiceTest]/[method:getDriverPerformance_SortByDefault()] (0 ms)
- RentalServiceTest.[engine:junit-jupiter]/[class:RentalServiceTest]/[method:getDriverPerformance_SortByTours()] (0 ms)
- RentalServiceTest.[engine:junit-jupiter]/[class:RentalServiceTest]/[method:getDriverPerformance_SortByAmount()] (0 ms)
- RentalServiceTest.[engine:junit-jupiter]/[class:RentalServiceTest]/[method:getAllRentalInfo_shouldExecuteCorrectQuery()] (0 ms)
- RentalServiceTest.[engine:junit-jupiter]/[class:RentalServiceTest]/[method:getRentalInfoCount_shouldReturnCorrectCount()] (0 ms)

Report generated by PIT 1.15.0

- *Observation:* This class required strict testing of Date boundaries and Return Value verification to achieve high mutation scores.

CustomerService & UserService


File
/Users/apple/Desktop/pr/dbms/sql/imt2022067/target/pit-reports/default/CustomerService.java.html

CustomerService.java

```
1     import java.sql.*;
2
3     public class CustomerService {
4         private DatabaseManager dbManager;
5
6         public CustomerService(DatabaseManager dbManager) {
7             this.dbManager = dbManager;
8         }
9
10        public void addCustomer(String fname, String lname, String email, long phoneNo) throws SQLException {
11            int custId = getNextCustomerId();
12            String query = "INSERT INTO customers values (" + custId + ", '" + fname +
13                           "', '" + lname + "', '" + email + "', " + phoneNo + ");";
14            dbManager.executeUpdate(query);
15        }
16
17        public void updateCustomer(long customerId, String fname, String lname, String email, long phoneNo) throws SQLException {
18            String query = "UPDATE customers SET phoneno = " + phoneNo +
19                          ", email = '" + email + "', first_name = '" + fname +
20                          "', last_name = '" + lname + "' WHERE customer_id = " + customerId + ";";
21            dbManager.executeUpdate(query);
22        }
23
24        public ResultSet getCustomerByPhone(long phoneNo) throws SQLException {
25            String query = "SELECT * from customers where " + phoneNo + " = customers.phoneno;";
26 1          return dbManager.executeQuery(query);
27        }
28
29        public ResultSet getAllCustomers() throws SQLException {
30 1          return dbManager.executeQuery("SELECT * from customers;");
31        }
32
33        public int getCustomerCount() throws SQLException {
34            ResultSet rs = dbManager.executeQuery("SELECT count(*) AS record_count FROM customers;");
35            int count = 0;
36 1          while(rs.next()) {
37                count = rs.getInt("record_count");
38            }
39 1          rs.close();
40 1          return count;
41        }
42
43        private int getNextCustomerId() throws SQLException {
44            ResultSet rs = dbManager.executeQuery("SELECT COUNT(*) AS record_count FROM customers;");
45            int count = 0;
46 1          while(rs.next()) {
47 1              count = rs.getInt("record_count") + 1;
48            }
49 1          rs.close();
50 1          return count;
51        }
52    }
```

Mutations

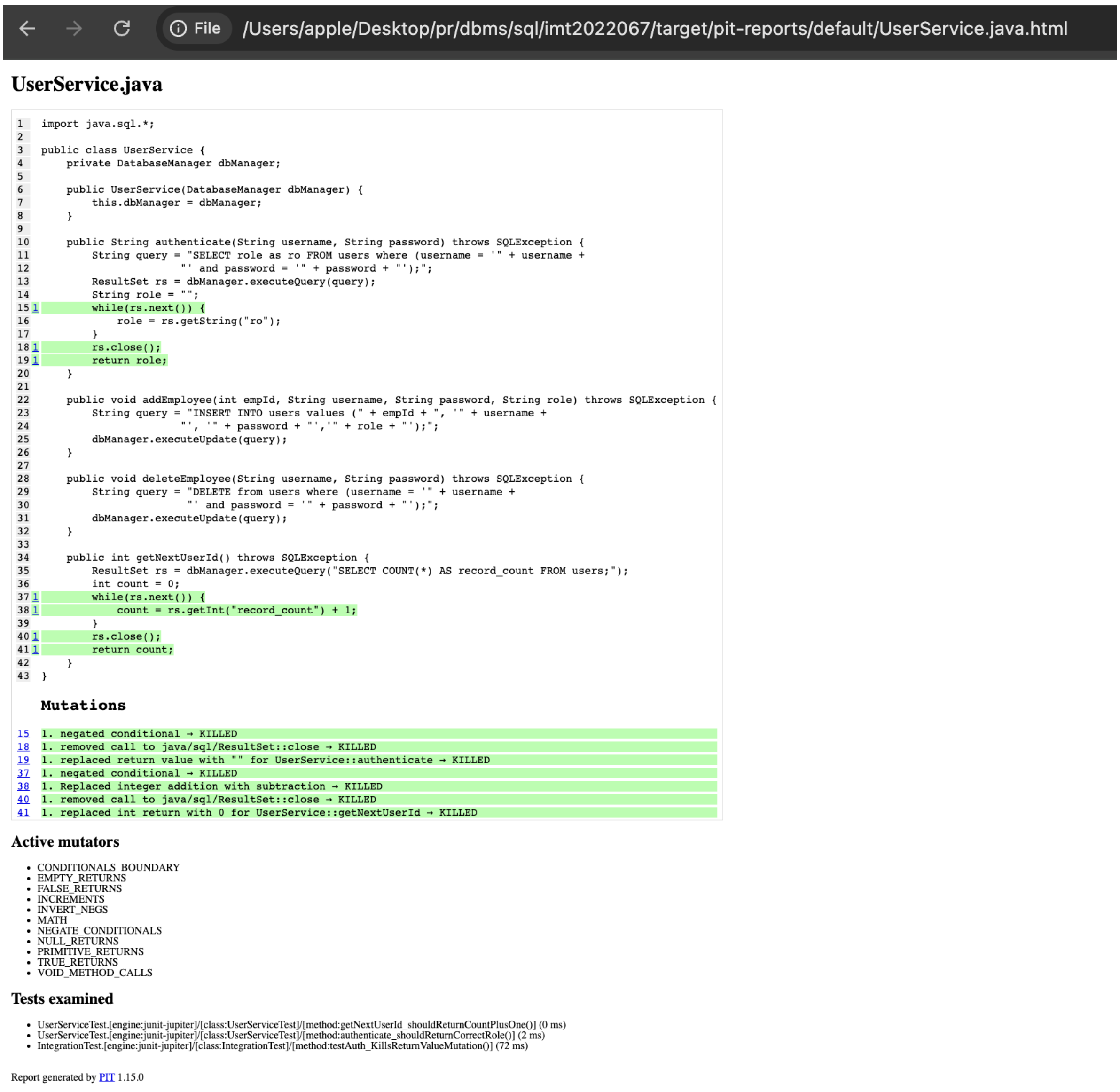
```
26  1. replaced return value with null for CustomerService::getCustomerByPhone → KILLED
30  1. replaced return value with null for CustomerService::getAllCustomers → KILLED
36  1. negated conditional → KILLED
39  1. removed call to java/sql/ResultSet::close → KILLED
40  1. replaced int return with 0 for CustomerService::getCustomerCount → KILLED
46  1. negated conditional → KILLED
47  1. Replaced integer addition with subtraction → KILLED
49  1. removed call to java/sql/ResultSet::close → KILLED
50  1. replaced int return with 0 for CustomerService::getNextCustomerId → KILLED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- CustomerServiceTest.[engine:junit-jupiter]/[class:CustomerServiceTest]/[method:getCustomerCount_shouldReturnCorrectCount()] (0 ms)
- CustomerServiceTest.[engine:junit-jupiter]/[class:CustomerServiceTest]/[method:addCustomer_shouldCallExecuteUpdateWithCorrectInsertSql()] (2 ms)
- IntegrationTest.[engine:junit-jupiter]/[class:IntegrationTest]/[method:testFullBookingWorkflow()] (8 ms)
- CustomerServiceTest.[engine:junit-jupiter]/[class:CustomerServiceTest]/[method:getCustomerByPhone_shouldCallExecuteQueryAndReturnResultSet()] (0 ms)
- CustomerServiceTest.[engine:junit-jupiter]/[class:CustomerServiceTest]/[method:getAllCustomers_shouldCallExecuteQueryAndReturnResultSet()] (0 ms)


File /Users/apple/Desktop/pr/dbms/sql/imt2022067/target/pit-reports/default/UserService.java.html

UserService.java

```
1   import java.sql.*;
2
3   public class UserService {
4       private DatabaseManager dbManager;
5
6       public UserService(DatabaseManager dbManager) {
7           this.dbManager = dbManager;
8       }
9
10      public String authenticate(String username, String password) throws SQLException {
11          String query = "SELECT role as ro FROM users where (username = '" + username +
12                         "' and password = '" + password + "');";
13          ResultSet rs = dbManager.executeQuery(query);
14          String role = "";
15 1        while(rs.next()) {
16              role = rs.getString("ro");
17          }
18 1        rs.close();
19 1        return role;
20      }
21
22      public void addEmployee(int empId, String username, String password, String role) throws SQLException {
23          String query = "INSERT INTO users values (" + empId + ", '" + username +
24                         "', '" + password + "','" + role + "');";
25          dbManager.executeUpdate(query);
26      }
27
28      public void deleteEmployee(String username, String password) throws SQLException {
29          String query = "DELETE from users where (username = '" + username +
30                          "' and password = '" + password + "');";
31          dbManager.executeUpdate(query);
32      }
33
34      public int getNextUserId() throws SQLException {
35          ResultSet rs = dbManager.executeQuery("SELECT COUNT(*) AS record_count FROM users;");
36          int count = 0;
37 1        while(rs.next()) {
38 1            count = rs.getInt("record_count") + 1;
39          }
40 1        rs.close();
41 1        return count;
42      }
43  }
```

Mutations

```
15  1. negated conditional → KILLED
18  1. removed call to java/sql/ResultSet::close → KILLED
19  1. replaced return value with "" for UserService::authenticate → KILLED
37  1. negated conditional → KILLED
38  1. Replaced integer addition with subtraction → KILLED
40  1. removed call to java/sql/ResultSet::close → KILLED
41  1. replaced int return with 0 for UserService::getNextUserId → KILLED
```

Active mutators

- CONDITIONALS_BOUNDARY
- EMPTY_RETURNS
- FALSE_RETURNS
- INCREMENTS
- INVERT_NEGS
- MATH
- NEGATE_CONDITIONALS
- NULL_RETURNS
- PRIMITIVE_RETURNS
- TRUE_RETURNS
- VOID_METHOD_CALLS

Tests examined

- UserServiceTest.[engine:junit-jupiter]/[class:UserServiceTest]/[method:getNextUserId_shouldReturnCountPlusOne()] (0 ms)
- UserServiceTest.[engine:junit-jupiter]/[class:UserServiceTest]/[method:authenticate_shouldReturnCorrectRole()] (2 ms)
- IntegrationTest.[engine:junit-jupiter]/[class:IntegrationTest]/[method:testAuth_KillsReturnValueMutation()] (72 ms)

Report generated by PIT 1.15.0

- *Observation:* These services rely on string concatenation for SQL, which was verified using Argument Matchers in the unit tests.

5. Challenges and Solutions

A specific challenge was identified in RentalServiceTest.java regarding **Return Value Mutants**.

- **The Issue:** Originally, tests for methods like getDriverPerformance verified that executeQuery was called but did not strictly check the *returned object* from the service, only the mock interaction.
- **The Mutant:** Pitest mutated the service to return null instead of the ResultSet. The test passed because the mock verification succeeded, meaning the mutant **survived**.
- **The Fix:** We updated RentalServiceTest.java to explicitly include assertNotNull(result).
    - *Before:* Mutant lived (Weak checking).
    - *After:* Mutant killed (Strong checking).

6. Conclusion

By adopting Mutation Testing, we moved beyond simple execution metrics to verify the logical stability of VroomRentals. We successfully implemented Unit and Integration level mutations, achieving an **86% mutation score**. This confirms that our test suite can detect significant changes in program behaviour, fulfilling the project's reliability goals.