

## JAVA BACKEND DELETE OPERATION REPORT

**Project Title:**Child Health Monitor System

**Name:**Divine AKISA

**Date:**12/02/2026

**Course\_Name:**Backend Using Java

### 1. Project Overview

This practical lab demonstrates the implementation of a DELETE operation in a Java backend application using JDBC and MySQL database connectivity. The chosen project topic is Child Health Monitor System.

The focus of this lab is to understand how Java connects to a database, executes SQL DELETE statements, and verifies the changes made in the database.

### 2. Database Process

#### 2.1 Database Creation

The database was created using the following SQL statement:

```
CREATE DATABASE chms_db;  
USE chms_db;
```

#### 2.2 Table Creation

Table: children

Purpose: Store child profiles registered by mothers

```
CREATE TABLE children (  
  child_id INT PRIMARY KEY AUTO_INCREMENT,  
  unique_profile_id VARCHAR(50) UNIQUE NOT NULL,  
  full_name VARCHAR(100) NOT NULL,  
  date_of_birth DATE NOT NULL,  
  gender ENUM('MALE', 'FEMALE', 'OTHER') NOT NULL,  
  birth_weight DECIMAL(5,2) COMMENT 'Weight at birth in kg',  
  birth_height DECIMAL(5,2) COMMENT 'Height at birth in cm',  
  blood_group VARCHAR(5),  
  mother_id INT NOT NULL,  
  father_name VARCHAR(100),  
  father_phone VARCHAR(20),  
  emergency_contact VARCHAR(20),  
  address TEXT,  
  medical_history TEXT COMMENT 'Pre-existing conditions, allergies',  
  is_active BOOLEAN DEFAULT TRUE,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
```

```

FOREIGN KEY (mother_id) REFERENCES users(user_id) ON DELETE RESTRICT,
INDEX idx_mother_id (mother_id),
INDEX idx_dob (date_of_birth),
INDEX idx_active (is_active)
) ENGINE=InnoDB;

```

The child\_id field uniquely identifies each children record.

## 2.3 Sample Data Insertion

Sample records were inserted for testing purposes:

```

INSERT INTO children (unique_profile_id, full_name, date_of_birth, gender, birth_weight,
birth_height, blood_group, mother_id, father_name, father_phone, emergency_contact, address,
medical_history) VALUES
('CH2023001234', 'Emma Wilson', '2023-03-15', 'FEMALE', 3.2, 50.0, 'O+', 5, 'Robert Wilson',
'+1234567899', '+1234567899', '123 Maple Street, Springfield, IL 62701', 'No known allergies'),
('CH2024002345', 'Oliver Wilson', '2024-06-20', 'MALE', 3.5, 52.0, 'O+', 5, 'Robert Wilson',
'+1234567899', '+1234567899', '123 Maple Street, Springfield, IL 62701', 'No known allergies'),

('CH2023003456', 'Sophia Brown', '2023-01-10', 'FEMALE', 3.0, 49.0, 'A+', 6, 'James Brown',
'+1234567810', '+1234567810', '456 Oak Avenue, Chicago, IL 60614', 'Lactose intolerant'),
('CH2024004567', 'Noah Brown', '2024-08-05', 'MALE', 3.8, 53.0, 'A+', 6, 'James Brown',
'+1234567810', '+1234567810', '456 Oak Avenue, Chicago, IL 60614', 'No known allergies'),
('CH2022005678', 'Isabella Davis', '2022-11-22', 'FEMALE', 3.3, 51.0, 'B+', 7, 'William Davis',
'+1234567811', '+1234567811', '789 Pine Road, Peoria, IL 61602', 'Peanut allergy'),
('CH2025006789', 'Liam Davis', '2025-01-12', 'MALE', 3.6, 51.5, 'B+', 7, 'William Davis',
'+1234567811', '+1234567811', '789 Pine Road, Peoria, IL 61602', 'No known allergies'),
('CH2023007890', 'Ava Miller', '2023-07-08', 'FEMALE', 3.1, 49.5, 'AB+', 8, 'Richard Miller',
'+1234567812', '+1234567812', '321 Elm Street, Rockford, IL 61101', 'Eczema'),
('CH2024008901', 'Ethan Moore', '2024-02-14', 'MALE', 3.4, 51.0, 'O-', 9, 'Charles Moore',
'+1234567813', '+1234567813', '654 Birch Lane, Naperville, IL 60540', 'No known allergies');

```

The screenshot shows the phpMyAdmin interface for a database named 'children'. The 'Structure' tab is selected, displaying a table with 8 columns: unique\_profile\_id, full\_name, date\_of\_birth, gender, birth\_weight, birth\_height, blood\_group, mother\_id, father\_name, father\_phone, emergency\_contact, address, and medical\_history. The table is using the InnoDB engine and utf8mb4\_unicode\_ci collation. Below the table structure, there is a 'Create new table' button and a form to create a new table with 4 columns.

Table	Action	Rows	Type	Collation	Size	Overhead
appointments		10	InnoDB	utf8mb4_unicode_ci	80.0 Kib	-
audit_logs		9	InnoDB	utf8mb4_unicode_ci	80.0 Kib	-
children		10	InnoDB	utf8mb4_unicode_ci	80.0 Kib	-
growth_alerts		2	InnoDB	utf8mb4_unicode_ci	96.0 Kib	-
health_records		33	InnoDB	utf8mb4_unicode_ci	80.0 Kib	-
notifications		15	InnoDB	utf8mb4_unicode_ci	96.0 Kib	-
users		10	InnoDB	utf8mb4_unicode_ci	80.0 Kib	-
vaccinations		20	InnoDB	utf8mb4_unicode_ci	64.0 Kib	-
<b>8 tables</b>	<b>Sum</b>	<b>117</b>	<b>InnoDB</b>	<b>utf8mb4_unicode_ci</b>	<b>656.0 Kib</b>	<b>0 B</b>

Database records before deletion(Mucyo Yannick)

Showing rows 0 - 9 (10 total. Query took 0.0007 seconds)

SELECT \* FROM "children"

Pushing [ Edit view ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]

Show all | Number of rows: 25 | Filter rows | Search this table | Sort by key: None

Extra options

	child_id	unique_profile_id	first_name	date_of_birth	gender	birth_weight <small>(pounds/kilograms)</small>	birth_height <small>(inches/cm)</small>	blood_group	mother_id	father_name	father_phone	emergency_contact	address	medical_history <small>(Phenylketonuria, allergies)</small>	is_active	created_at	updated_at
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	1	CH0203001234	Emma Wilson	2024-03-15	FEMALE	3.20	58.00	O+	5	Robert Wilson	+1234567899	+1234567899	123 Maple Street, Springfield, IL 62701	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2	CH0204002345	Oliver Wilson	2024-06-29	MALE	3.50	62.00	O+	5	Robert Wilson	+1234567899	+1234567899	123 Maple Street, Springfield, IL 62701	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	3	CH0203003456	Sophia Brown	2023-01-19	FEMALE	3.80	49.00	A+	6	James Brown	+1234567810	+1234567810	456 Oak Avenue, Chicago, IL 60614	Lactose intolerant	1	2026-02-12 11:21:42	2026-02-12 11:21:42
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	4	CH0204004567	Noah Brown	2024-05-05	MALE	3.80	53.00	A+	6	James Brown	+1234567810	+1234567810	456 Oak Avenue, Chicago, IL 60614	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	5	CH0202005678	Isabella Davis	2022-11-22	FEMALE	3.30	51.00	B+	7	William Davis	+1234567811	+1234567811	789 Pine Road, Peoria, IL 61602	Peanut allergy	1	2026-02-12 11:21:42	2026-02-12 11:21:42
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	6	CH0205006789	Liam Davis	2025-01-12	MALE	3.60	51.50	B+	7	William Davis	+1234567811	+1234567811	789 Pine Road, Peoria, IL 61602	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	7	CH0203007890	Ava Miller	2023-07-08	FEMALE	3.10	49.50	AB+	8	Richard Miller	+1234567812	+1234567812	321 Elm Street, Rockford, IL 61101	Eczema	1	2026-02-12 11:21:42	2026-02-12 11:21:42
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	8	CH0204008901	Ethan Moore	2024-02-14	MALE	3.40	51.00	O-	9	Charles Moore	+1234567813	+1234567813	604 Birch Lane, Naperville, IL 60540	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	9	CH0206009012	Mucyo Abel	2026-02-04	FEMALE	1.20	6.50	A-	10	Mandi	0788888884	0788884032	Koukoro	None	1	2026-02-12 13:00:36	2026-02-12 13:00:36
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	10	CH0207009044	Mucyo Yanick	2026-02-11	FEMALE	2.10	2.30	B+	10	Mandi	0788888884	0788884032	Koukoro	hh	1	2026-02-12 13:01:55	2026-02-12 13:01:55

Show all | Number of rows: 25 | Filter rows | Search this table | Sort by key: None

Query results operations

Print | Copy to clipboard | Export | Display chart | Create view

Bookmark this SQL query

Label:  ☐ Let every user access this bookmark

Bookmark this SQL query

## Before Delete Children

CHMS

Search by Date | Akisa Divine | Logout

### Welcome back, Akisa Divine!

Here's an overview of your children's health

2

Total Children

0

Upcoming Appointments

0

New Notifications

#### My Children

+ Add Child

Mucyo Yanick

ID: CH0206709044

DOB: 2026-02-11

Blood Group: B+

Birth Weight: 2.1 kg

[View Details](#) [Edit](#)

Mucyo Abel

ID: CH0206009034

DOB: 2026-02-04

Blood Group: A-

Birth Weight: 1.2 kg

[View Details](#) [Edit](#)

#### Recent Notifications

No new notifications

#### Upcoming Appointments

No upcoming appointments

Mucyo Yanick

CH2026706044

Basic Information

Full Name: Mucyo Yanick

Date of Birth: 2026-02-11

Age: 0 years, 0 months (0 months total)

Gender: FEMALE

Blood Group: B+

Birth Information

Birth Weight: 2.1 kg

Birth Height: 2.3 cm

Profile ID: CH2026706044

Parent Information

Mother: Akisa Divine

Mother's Phone: 0792502568

Father: Mandi

Father's Phone: 0788888804

Contact Information

Emergency Contact: 0788864532

Address:  
Kicukiro

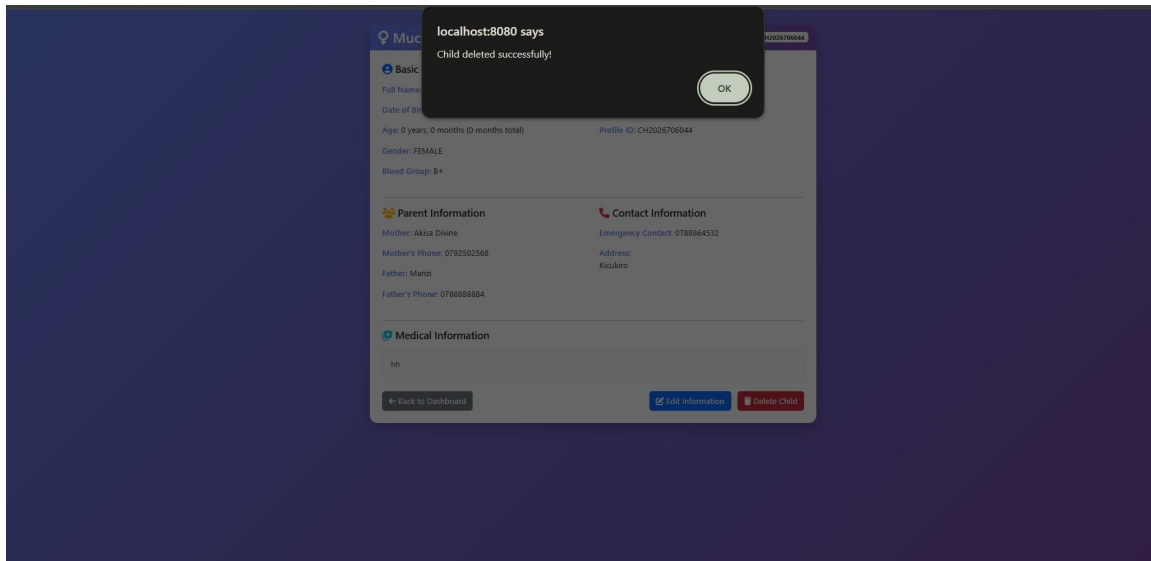
Medical Information

hh

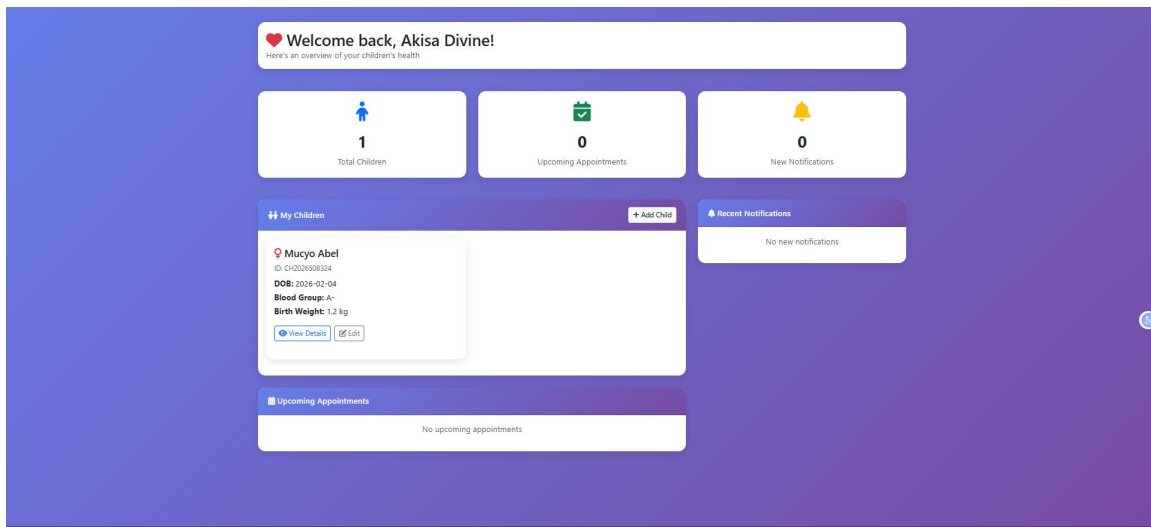
← Back to Dashboard

Edit InformationDelete Child

The image is a screenshot of a web application interface. In the foreground, a dark grey modal dialog box is centered, displaying a confirmation message: "Are you sure you want to delete Muoyo Yanick? This action cannot be undone!". Below the message are two buttons: "OK" and "Cancel". The background is a blurred view of a user profile page for "Muoyo Yanick". The page has a header with a location pin icon and the name "Muoyo Yanick". Below the header, there are sections for "Basic Information", "Parent Information", "Contact Information", and "Medical Information". The "Basic Information" section shows fields for Full Name, Date of Birth, Age, Gender, and Blood Group. The "Parent Information" section shows fields for Mother's Name, Mother's Phone, Father's Name, and Father's Phone. The "Contact Information" section shows fields for Emergency Contact and Address. The "Medical Information" section shows a field for Height. At the bottom of the page, there are two buttons: "Back to Dashboard" and "Edit Information".



After Delete Operation



Before executing the DELETE operation, the mother had 2 children but now remain with one children

## 2.4 Database State After DELETE

After executing the DELETE operation, one selected childern record was removed from the database. The table then displayed the remaining records only.

**Database records after deletion**

Showing rows 0 - 9 (9 total. Query took 0.0009 seconds)

**SELECT \* FROM "children"**

Profiling [Edit view] [SQL] [Explain SQL] [Create PHP code] [Refresh]

Show all Number of rows: 25 Filter rows Search this table Sort by key: None

child_id	unique_profile_id	first_name	date_of_birth	gender	birth_weight (kg)	birth_height (cm)	blood_group	mother_id	father_name	father_phone	emergency_contact	address	medical_history (prevalent/allergies/conditions)	is_active	created_at	updated_at
1	CH2023001234	Emma Wilson	2023-03-15	FEMALE	3.20	50.00	O+	5	Robert Wilson	+1234567890	+1234567890	123 Maple Street, Springfield, IL 62701	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
2	CH2024002345	Oliver Wilson	2024-06-20	MALE	3.50	52.00	O+	5	Robert Wilson	+1234567890	+1234567890	123 Maple Street, Springfield, IL 62701	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
3	CH2023003456	Sophia Brown	2023-01-10	FEMALE	3.00	49.00	A+	6	James Brown	+1234567810	+1234567810	456 Oak Avenue, Chicago, IL 60614	Lactose intolerant	1	2026-02-12 11:21:42	2026-02-12 11:21:42
4	CH2024004567	Noah Brown	2024-08-05	MALE	3.90	53.00	A+	6	James Brown	+1234567810	+1234567810	456 Oak Avenue, Chicago, IL 60614	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
5	CH2022005678	Isabella Davis	2022-11-22	FEMALE	3.30	51.00	B+	7	William Davis	+1234567811	+1234567811	789 Pine Road, Peoria, IL 61602	Peanut allergy	1	2026-02-12 11:21:42	2026-02-12 11:21:42
6	CH2025006789	Liam Davis	2025-01-12	MALE	3.60	51.50	B+	7	William Davis	+1234567811	+1234567811	789 Pine Road, Peoria, IL 61602	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
7	CH2023007890	Ava Miller	2023-07-08	FEMALE	3.10	49.50	A-	8	Rhonda Miller	+1234567812	+1234567812	321 Elm Street, Rockford, IL 61101	Eczema	1	2026-02-12 11:21:42	2026-02-12 11:21:42
8	CH2024008901	Ethan Moore	2024-02-14	MALE	3.40	51.00	O-	9	Charles Moore	+1234567813	+1234567813	654 Birch Lane, Naperville, IL 60563	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
9	CH2025009012	Mia Taylor	2025-02-04	FEMALE	1.20	0.50	A-	10	Marc Taylor	0765000004	0765000432	Kankakee	None	1	2026-02-12 13:00:36	2026-02-12 13:00:36

Check all With selected Edit Copy Delete Export

Show all Number of rows: 25 Filter rows Search this table Sort by key: None

Query results operations

Print Copy to clipboard Export Display chart Create view

Bookmark this SQL query

Label:  ☐ Let every user access this bookmark

Bookmark this SQL query

### 3. Java Implementation

#### 3.1 Database Connection Code

The following code establishes a connection between Java and MySQL database:

```

1 // main / pom / src / test / java / DatabaseConnection.java
2
3 import java.util.Properties;
4
5 /**
6  * Database Connection Manager using Apache Commons DBCP Connection Pooling
7  * This class provides thread-safe database connections for the OOPS application
8  */
9 public class DatabaseConnection {
10
11     private static final Logger logger = LoggerFactory.getLogger(DatabaseConnection.class);
12     private static BasicDataSource dataSource;
13
14     // Static block to initialize the connection pool
15     static {
16         try {
17             initializeDataSource();
18             logger.info("Database connection pool initialized successfully");
19         } catch (Exception e) {
20             logger.error("Failed to initialize database connection pool", e);
21             throw new RuntimeException("Failed to initialize database connection pool: " + e.getMessage());
22         }
23     }
24
25     /**
26      * Initialize the database connection pool
27      */
28     private static void initializeDataSource() throws IOException {
29         Properties props = loadDatabaseProperties();
30
31         dataSource = new BasicDataSource();
32
33         // Basic connection properties
34         dataSource.setDriverClassName(props.getProperty("db.driver"));
35         dataSource.setUrl(props.getProperty("db.url"));
36         dataSource.setUsername(props.getProperty("db.username"));
37         dataSource.setPassword(props.getProperty("db.password"));
38
39         // Connection pool configuration
40         dataSource.setInitialSize(Integer.parseInt(props.getProperty("db.initialSize", "5")));
41         dataSource.setMaxTotal(Integer.parseInt(props.getProperty("db.maxTotal", "10")));
42         dataSource.setMaxIdle(Integer.parseInt(props.getProperty("db.maxIdle", "5")));
43         dataSource.setMinIdle(Integer.parseInt(props.getProperty("db.minIdle", "2")));
44         dataSource.setMaxWaitMillis(Long.parseLong(props.getProperty("db.maxWaitMillis", "10000")));
45
46         // Connection validation
47         dataSource.setValidationQuery("SELECT 1");
48         dataSource.setTestOnBorrow(true);
49         dataSource.setTestWhileIdle(true);
50         dataSource.setTimeBetweenEvictionRunsMillis(30000);
51
52         // Connection timeout settings
53         dataSource.setRemoveAbandonedOnBorrow(true);
54         dataSource.setRemoveAbandonedTimeout(60);
55         dataSource.setLogAbandoned(true);
56     }
57
58     // Load database properties from a file
59     private static Properties loadDatabaseProperties() throws IOException {
60         Properties props = new Properties();
61         props.load(new FileInputStream("src/main/resources/db.properties"));
62         return props;
63     }
64 }

```

```

src> main> java> com> chms> util> J DatabaseConnection.java> {} com.chms.util
17 public class DatabaseConnection {
18     private static void initializeDataSource() throws IOException {
19         dataSource.setMaxActive(Integer.parseInt(props.getProperty("db.maxactive", "3"));
20         dataSource.setMinIdle(Integer.parseInt(props.getProperty("db.minidle", "2"));
21         dataSource.setMaxWaitMillis(Long.parseLong(props.getProperty("db.maxwaitmillis", "10000")));
22     }
23
24     // Connection validation
25     dataSource.setValidationQuery("SELECT 1");
26     dataSource.setTestOnBorrow(true);
27     dataSource.setTestWhileIdle(true);
28     dataSource.setTestOnBorrowMillis(30000);
29
30     // Connection timeout settings
31     dataSource.setRemoveAbandonedOnBorrow(true);
32     dataSource.setRemoveAbandonedTimeout(60);
33     dataSource.setLogAbandoned(true);
34 }
35
36 /**
37  * Load database properties from configuration file
38  */
39 private static Properties loadDatabaseProperties() throws IOException {
40     Properties props = new Properties();
41
42     try (InputStream input = DatabaseConnection.class.getClassLoader()
43         .getResourceAsStream("database.properties")) {
44         if (input == null) {
45             throw new IOException("Unable to find database.properties file");
46         }
47         props.load(input);
48         logger.info("Database properties loaded successfully");
49     } catch (IOException e) {
50         logger.error("Error loading database properties", e);
51         throw e;
52     }
53
54     return props;
55 }
56
57 /**
58  * Get a connection from the pool
59  * @return Connection object
60  * @throws SQLException if unable to get connection
61  */
62 public static Connection getConnection() throws SQLException {
63     if (dataSource == null) {
64         throw new SQLException("DataSource is not initialized");
65     }
66
67     Connection conn = dataSource.getConnection();
68     logger.debug("Connection obtained from pool. Active connections: {}", dataSource.getNumActive());
69     return conn;
70 }

```

```

71 /**
72  * Close the entire connection pool (should be called on application shutdown)
73  */
74 public static void closePool() {
75     if (dataSource != null) {
76         try {
77             dataSource.close();
78             logger.info("Database connection pool closed successfully");
79         } catch (SQLException e) {
80             logger.error("Error closing connection pool", e);
81         }
82     }
83 }
84
85 /**
86  * Get connection pool statistics
87  * @return String containing pool statistics
88  */
89 public static String getPoolStats() {
90     if (dataSource == null) {
91         return "DataSource not initialized";
92     }
93
94     return String.format(
95         "Connection Pool Stats - Active: %d, Idle: %d, Max: %d",
96         dataSource.getNumActive(),
97         dataSource.getNumIdle(),
98         dataSource.getMaxTotal());
99 }
100
101 /**
102  * Test database connection
103  * @return true if connection successful, false otherwise
104  */
105 public static boolean testConnection() {
106     try (Connection conn = getConnection()) {
107         boolean isValid = conn.isValid(5);
108         if (isValid) {
109             logger.info("Database connection test successful");
110         } else {
111             logger.warn("Database connection test failed");
112         }
113         return isValid;
114     } catch (SQLException e) {
115         logger.error("Database connection test failed", e);
116         return false;
117     }
118 }

```

## 3.2 DELETE Operation Code

The following Java code is used to delete a Children record by child\_id:

```

Delete a child record
DELETE FROM children WHERE child_id = ?;
Delete a user record
DELETE FROM users WHERE user_id = ?;

String sql = "DELETE FROM children WHERE child_id = ?";

try (Connection conn = DatabaseConnection.getConnection();
    PreparedStatement pstmt = conn.prepareStatement(sql)) {

    pstmt.setInt(1, childId);
    int affectedRows = pstmt.executeUpdate();

```

```

        if (affectedRows > 0) {
            logger.info("Child deleted successfully: {}", childId);
            return true;
        }
    } catch (SQLException e) {
        logger.error("Error deleting child: {}", childId, e);
    }
    return false;
}
}

```

```

public boolean deleteUser(int userId) {
    String sql = "DELETE FROM users WHERE user_id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setInt(1, userId);
        int affectedRows = pstmt.executeUpdate();

        if (affectedRows > 0) {
            logger.info("User deleted permanently: {}", userId);
            return true;
        }
    } catch (SQLException e) {
        logger.error("Error deleting user: {}", userId, e);
    }
    return false;
}
}

```

The DELETE operation uses a PreparedStatement with a parameterized query to safely remove a record based on its ID. The executeUpdate() method returns the number of rows affected, which confirms whether the deletion was successful.

When a mother provides a child\_id, the application processes the request, executes the DELETE statement, and displays a confirmation message based on the result.

#### 4. Conclusion

In this practical lab, the DELETE operation was successfully implemented in a Java backend application using JDBC. The project demonstrated how Java interacts with a MySQL database, how SQL DELETE statements are executed, and how database changes are verified before and after deletion.

Challenges faced included ensuring correct database connection configuration and handling SQL exceptions properly. These challenges were solved by verifying connection parameters and using proper try-catch blocks.

This lab enhanced understanding of backend database operations and strengthened practical skills in Java database connectivity.