

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');

```

Database records before deletion(Mucyo Yannick)

Servers 127.0.0.1 Database chms_db Table children

Browse Structure SQL Search Insert Export Import Privileges Operations Tracking Triggers

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

SELECT * FROM `children`

Profiling | Edit index | [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	full_name	date_of_birth	gender	birth_weight	birth_height	blood_group	mother_id	father_name	father_phone	emergency_contact	address	medical_history	is_active	created_at	updated_at
1	CH2023001234	Emma Wilson	2023-01-15	FEMALE	3.20	50.00	O+	5	Robert Wilson	+1234567899	+1234567899	123 Maple Street, Springfield, IL 62701	No known allergies	1	2023-02-12 11:21:42	2023-02-12 11:21:42
2	CH2024002345	Oliver Wilson	2024-06-20	MALE	3.50	52.00	O+	5	Robert Wilson	+1234567899	+1234567899	123 Maple Street, Springfield, IL 62701	No known allergies	1	2023-02-12 11:21:42	2023-02-12 11:21:42
3	CH2023003456	Sophia Brown	2023-01-10	FEMALE	3.00	49.00	A+	8	James Brown	+1234567810	+1234567810	456 Oak Avenue, Chicago, IL 60615	Lactose intolerant	1	2023-02-12 11:21:42	2023-02-12 11:21:42
4	CH2023004567	Noah Brown	2023-06-05	MALE	3.80	53.00	A+	8	James Brown	+1234567810	+1234567810	456 Oak Avenue, Chicago, IL 60615	No known allergies	1	2023-02-12 11:21:42	2023-02-12 11:21:42
5	CH2023005678	Isabella Davis	2023-03-02	FEMALE	3.30	51.50	B+	7	William Davis	+1234567811	+1234567811	789 Pine Road, Peoria, IL 61602	Peanut allergy	1	2023-02-12 11:21:42	2023-02-12 11:21:42
6	CH2023006789	Liam Davis	2023-05-12	MALE	3.00	52.50	B+	7	William Davis	+1234567811	+1234567811	789 Pine Road, Peoria, IL 61602	No known allergies	1	2023-02-12 11:21:42	2023-02-12 11:21:42
7	CH2023007890	Ava Miller	2023-07-06	FEMALE	3.10	49.50	AB+	9	Richard Miller	+1234567812	+1234567812	521 Elm Street, Rockford, IL 61101	Eczema	1	2023-02-12 11:21:42	2023-02-12 11:21:42
8	CH2024008901	Ethan Moore	2024-03-14	MALE	3.40	51.00	O-	9	Charles Moore	+1234567813	+1234567813	654 Birch Lane, Naperville, IL 60540	No known allergies	1	2023-02-12 11:21:42	2023-02-12 11:21:42
9	CH2023009002	Macy Abel	2023-02-04	FEMALE	1.20	55.50	A-	10	Marcia Abel	0789886452	0789886452	Kukuleo	None	1	2023-02-12 13:00:36	2023-02-12 13:00:36
10	CH2023070044	Mucyo Yanick	2023-02-11	FEMALE	2.10	2.30	B+	10	Marcia Abel	0789886454	0789886454	Kukuleo	None	1	2023-02-12 13:01:55	2023-02-12 13:01:55

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

Console

Before Delete Children

CHMS

Welcome back, Akisa Divine!

Here's an overview of your children's health



2

Total Children



0

Upcoming Appointments



0

New Notifications

My Children

Mucyo Yanick
ID: CH2023070044
DOB: 2023-02-11
Blood Group: B+
Birth Weight: 2.1 kg

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

Mucyo Abel
ID: CH2024009002
DOB: 2024-03-14
Blood Group: A-
Birth Weight: 1.2 kg

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

Recent Notifications

No new notifications

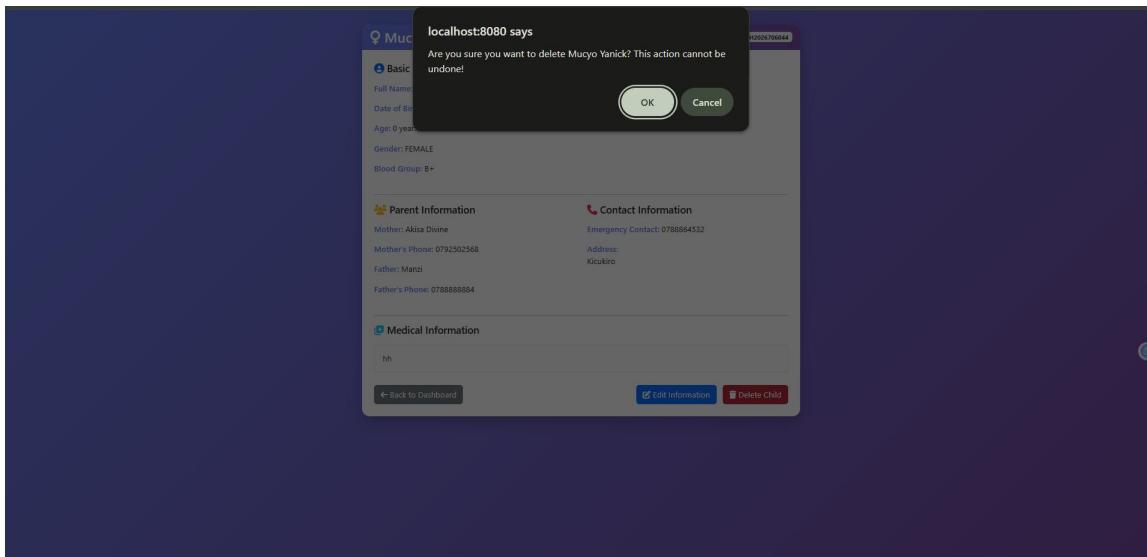
Upcoming Appointments

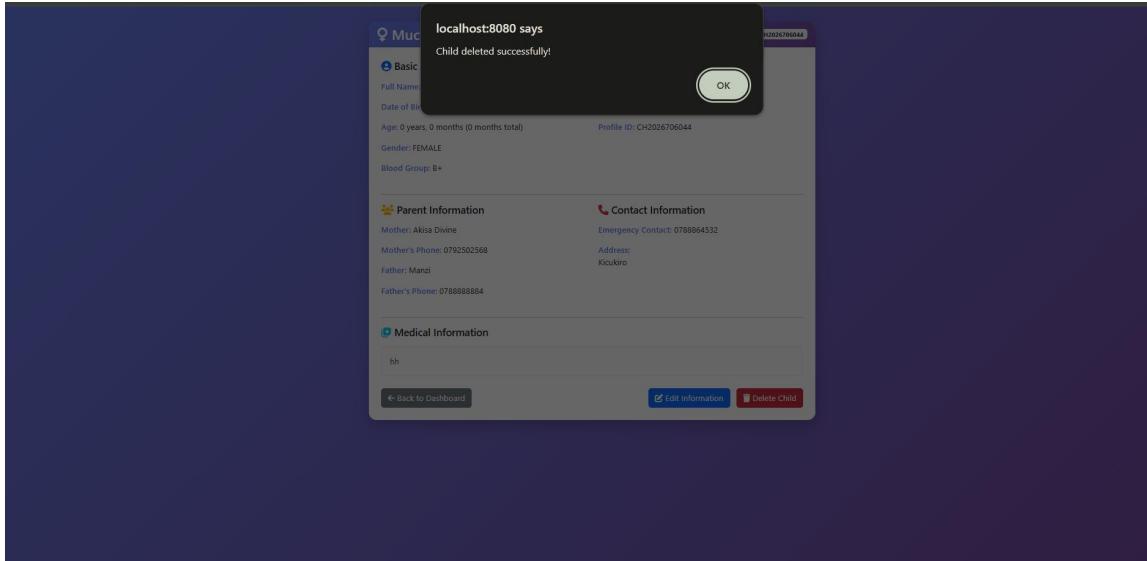
No upcoming appointments

 Mucyo Yanick CH2026706044

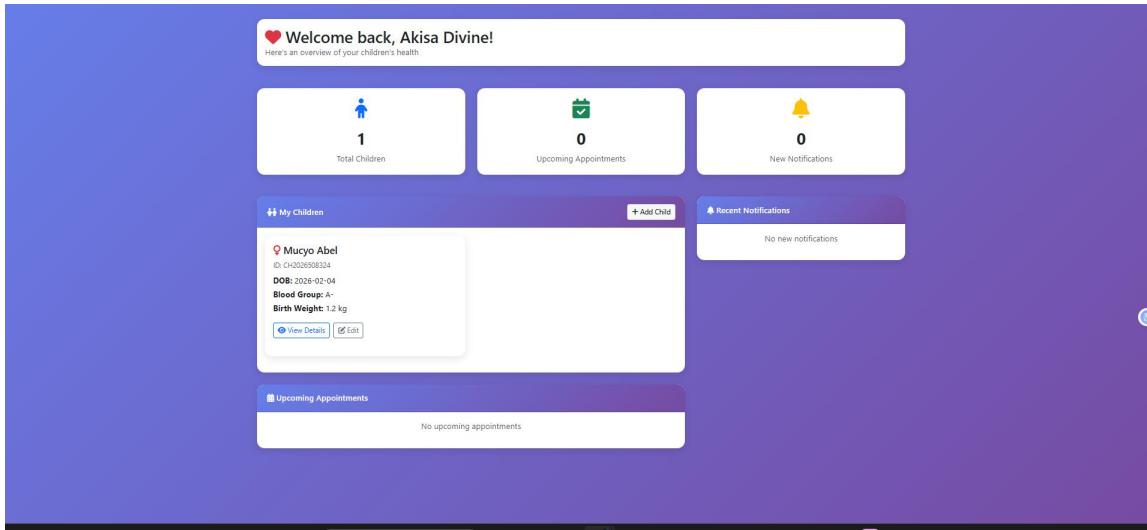
Basic Information Full Name: Mucyo Yanick Date of Birth: 2026-02-11 Age: 0 years, 0 months (0 months total) Gender: FEMALE Blood Group: B+	Birth Information Birth Weight: 2.1 kg Birth Height: 2.3 cm Profile ID: CH2026706044
Parent Information Mother: Akisa Divine Mother's Phone: 0792502568 Father: Mandi Father's Phone: 0788888864	Contact Information Emergency Contact: 0788864532 Address: Kicukiro
Medical Information hh	

[← Back to Dashboard](#) [Edit information](#) [Delete Child](#)





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

child_id	unique_profile_id	full_name	date_of_birth	gender	birth_weight	birth_height	blood_group	mother_id	father_id	father_phone	emergency_contact	address	medical_history	is_active	created_at	updated_at
1	CH20230301234	Emma Wilson	2023-03-15	FEMALE	3.00	20.00	O+	5	Robert Wilson	+1234567890	123 Maple Street, Springfield, IL 62701	No known allergies	1	2023-03-12 11:21:42	2026-02-10 11:21:42	
2	CH20230302345	Oliver Wilson	2023-06-20	MALE	3.50	21.00	O+	5	Robert Wilson	+1234567890	123 Maple Street, Springfield, IL 62701	No known allergies	1	2023-03-12 11:21:42	2026-02-10 11:21:42	
3	CH20230301456	Sophie Brown	2023-01-10	FEMALE	3.00	21.00	A+	6	James Brown	+1234567810	456 Oak Avenue, Chicago, IL 60614	Lactose intolerant	1	2023-03-12 11:21:42	2026-02-10 11:21:42	
4	CH202303004567	Noah Brown	2023-08-05	MALE	3.00	21.00	A+	6	James Brown	+1234567810	456 Oak Avenue, Chicago, IL 60614	No known allergies	1	2023-03-12 11:21:42	2026-02-10 11:21:42	
5	CH20230901789	Isabella Davis	2023-11-22	FEMALE	3.30	21.00	B+	7	William Davis	+1234567811	789 Pine Road, Peoria, IL 61602	Peanut allergy	1	2023-03-12 11:21:42	2026-02-10 11:21:42	
6	CH20230901789	Liam Davis	2023-01-12	MALE	3.60	21.50	B+	7	William Davis	+1234567811	789 Pine Road, Peoria, IL 61602	No known allergies	1	2023-03-12 11:21:42	2026-02-10 11:21:42	
7	CH202307089	Ava Miller	2023-07-08	FEMALE	3.10	20.50	A+	8	Richard Miller	+1234567812	371 Elm Street, Rockford, IL 61101	Eczema	1	2023-03-12 11:21:42	2026-02-10 11:21:42	
8	CH20230409091	Ethan Moore	2023-02-14	MALE	3.40	21.00	O-	9	Charles Moore	+1234567813	654 Birch Lane, Naperville, IL 60540	No known allergies	1	2023-03-12 11:21:42	2026-02-10 11:21:42	
9	CH2023050324	Mugay Abel	2026-02-04	FEMALE	1.20	0.50	A-	10	Meng	0700000084	0700000084	Kreatiro	None	1	2026-02-12 13:00:36	2026-02-12 13:00:36

3. Java Implementation

3.1 Database Connection Code

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

```
rc> main > java -cp com> util > J DatabaseConnection.java > [com.chris.util
1
2  import java.util.Properties;
3
4
5 /**
6 * Database Connection Manager using Apache Commons DBCP Connection Pooling
7 * This class provides thread-safe database connections for the CHRS 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.getMessage());
21         }
22     }
23
24     /**
25      * Initialize the database connection pool
26     */
27     private static void initializeDataSource() throws IOException {
28         Properties props = loadDatabaseProperties();
29
30         dataSource = new BasicDataSource();
31
32         // Basic connection properties
33         dataSource.setDriverClassName(props.getProperty("db.driver"));
34         dataSource.setUrl(props.getProperty("db.url"));
35         dataSource.setUsername(props.getProperty("db.username"));
36         dataSource.setPassword(props.getProperty("db.password"));
37
38         // Connection pool configuration
39         dataSource.setInitialSize(Integer.parseInt(props.getProperty("db.initialSize", "0")));
40         dataSource.setMaxTotal(Integer.parseInt(props.getProperty("db.maxTotal", "10")));
41         dataSource.setMinIdle(Integer.parseInt(props.getProperty("db.minIdle", "5")));
42         dataSource.setMaxIdle(Integer.parseInt(props.getProperty("db.maxIdle", "2")));
43         dataSource.setRemoveAbandoned(true);
44         dataSource.setRemoveAbandonedTimeout(30000);
45
46         // Connection timeout settings
47         dataSource.setQueryTimeout(1);
48         dataSource.setTestOnBorrow(true);
49         dataSource.setTestOnReturn(true);
50         dataSource.setValidationQuery("SELECT 1");
51
52         // Connection timeout settings
53         dataSource.setRemoveAbandoned(true);
54         dataSource.setRemoveAbandonedTimeout(60);
55         dataSource.setPing(true);
56
57     }
58 }
```

```

src/main/java/com/dms/dto/DatabaseConnection.java > {} com.chmssutl
17  public class DatabaseConnection {
18
19      private static void initializeDataSource() throws IOException {
20          Properties prop = new Properties();
21          prop.load(new FileInputStream("src/main/resources/database.properties"));
22          dataSource.setUrl(prop.getProperty("db.url"));
23          dataSource.setDriverClassName(prop.getProperty("db.driverName"));
24          dataSource.setUsername(prop.getProperty("db.username", "root"));
25          dataSource.setPassword(prop.getProperty("db.password", "root"));
26
27          // Connection validation
28          dataSource.setValidationQuery("SELECT 1");
29          dataSource.setTestOnBorrow(true);
30          dataSource.setTestOnBleedle(true);
31          dataSource.setValidateConnectionTimeout(30000);
32
33          // Connection timeout settings
34          dataSource.setConnectionTimeout(10000);
35          dataSource.setRemoveAbandoned(true);
36          dataSource.setLogAbandoned(true);
37
38      }
39
40      /**
41      * Load database properties from configuration file
42      */
43      private static Properties loadDatabaseProperties() throws IOException {
44          Properties props = new Properties();
45
46          try (InputStream input = DatabaseConnection.class.getClassLoader().getResourceAsStream("database.properties")) {
47
48              if (input == null) {
49                  throw new IOException("Unable to find database.properties file");
50              }
51
52              props.load(input);
53              logger.info("Database properties loaded successfully");
54
55          } catch (IOException e) {
56              logger.error("Error loading database properties", e);
57              throw e;
58          }
59
60          return props;
61      }
62
63      /**
64      * Get a connection from the pool
65      * @return Connection object
66      * @throws SQLException if unable to get connection
67      */
68      public static Connection getConnection() throws SQLException {
69          if (dataSource == null) {
70              throw new SQLException("DataSource is not initialized");
71          }
72
73          Connection conn = dataSource.getConnection();
74          logger.debug("Connection obtained from pool. Active connections: {}", dataSource.getNumActive());
75
76          return conn;
77      }
78
79      /**
80      * Close the entire connection pool (should be called on application shutdown)
81      */
82      public static void closePool() {
83          if (dataSource != null) {
84              try {
85                  dataSource.close();
86                  logger.info("Database connection pool closed successfully");
87              } catch (SQLException e) {
88                  logger.error("Error closing connection pool", e);
89              }
90          }
91
92      }
93
94      /**
95      * Get connection pool statistics
96      * @return String containing pool statistics
97      */
98      public static String getPoolStats() {
99          if (dataSource == null) {
100              return "DataSource not initialized";
101          }
102
103          return String.format(
104              "Connection Pool Stats - Active: %d, Idle: %d, Max: %d",
105              dataSource.getNumActive(),
106              dataSource.getNumIdle(),
107              dataSource.getMaxTotal()
108          );
109      }
110
111      /**
112      * Test database connection
113      * @return true if connection successful, false otherwise
114      */
115      public static boolean testConnection() {
116          try (Connection conn = getConnection()) {
117              boolean isValid = conn.isValid(0);
118              if (isValid) {
119                  logger.info("Database connection test successful");
120              } else {
121                  logger.warn("Database connection test failed");
122              }
123              return isValid;
124          } catch (SQLException e) {
125              logger.error("Database connection test failed", e);
126          }
127      }
128  }

```

```

/*
 * Close the entire connection pool (should be called on application shutdown)
 */
public static void closePool() {
    if (dataSource != null) {
        try {
            dataSource.close();
            logger.info("Database connection pool closed successfully");
        } catch (SQLException e) {
            logger.error("Error closing connection pool", e);
        }
    }
}

/**
 * Get connection pool statistics
 * @return String containing pool statistics
 */
public static String getPoolStats() {
    if (dataSource == null) {
        return "DataSource not initialized";
    }

    return String.format(
        "Connection Pool Stats - Active: %d, Idle: %d, Max: %d",
        dataSource.getNumActive(),
        dataSource.getNumIdle(),
        dataSource.getMaxTotal()
    );
}

/**
 * Test database connection
 * @return true if connection successful, false otherwise
 */
public static boolean testConnection() {
    try (Connection conn = getConnection()) {
        boolean isValid = conn.isValid(0);
        if (isValid) {
            logger.info("Database connection test successful");
        } else {
            logger.warn("Database connection test failed");
        }
        return isValid;
    } catch (SQLException e) {
        logger.error("Database connection test failed", e);
    }
}

```

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.