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

```sql
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,
```

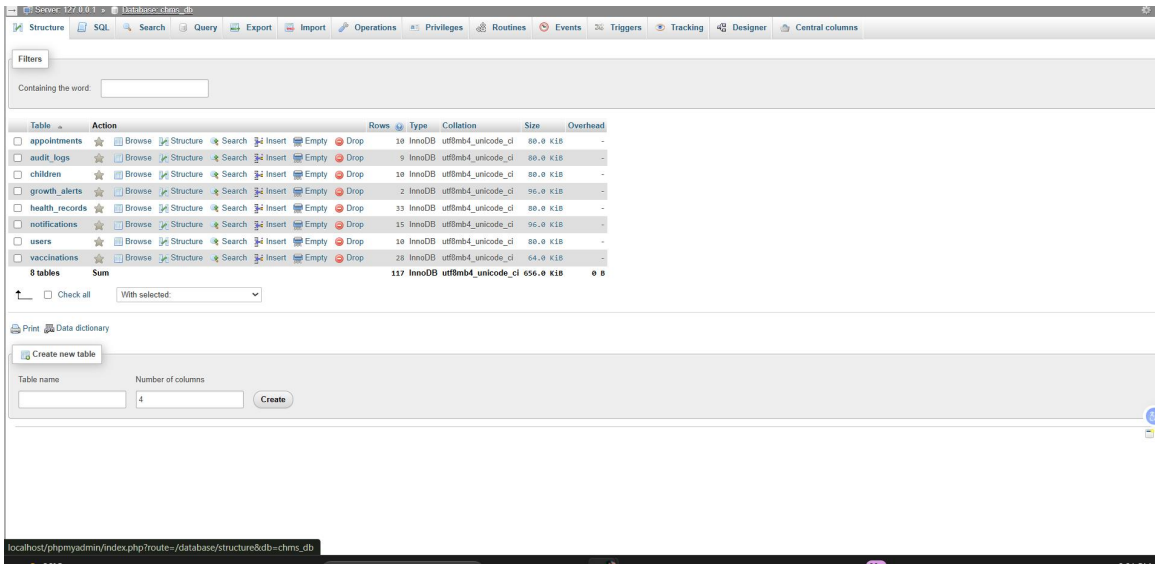
```
    FOREIGN KEY (mother_id) REFERENCES users(user_id) ON DELETE RESTRICT,
    INDEX idx_mother_id (mother_id),
    INDEX idx_dob (date_of_birth),
    INDEX idx_active (is_active)
) ENGINE=InnoDB;
```

The child_id field uniquely identifies each children record.

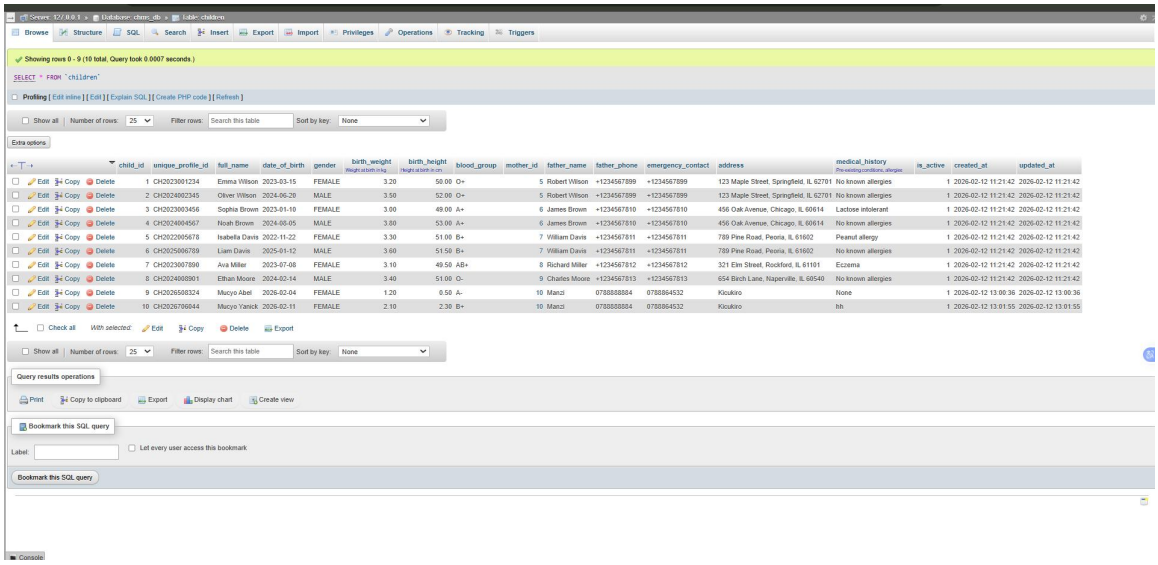
2.3 Sample Data Insertion

Sample records were inserted for testing purposes:

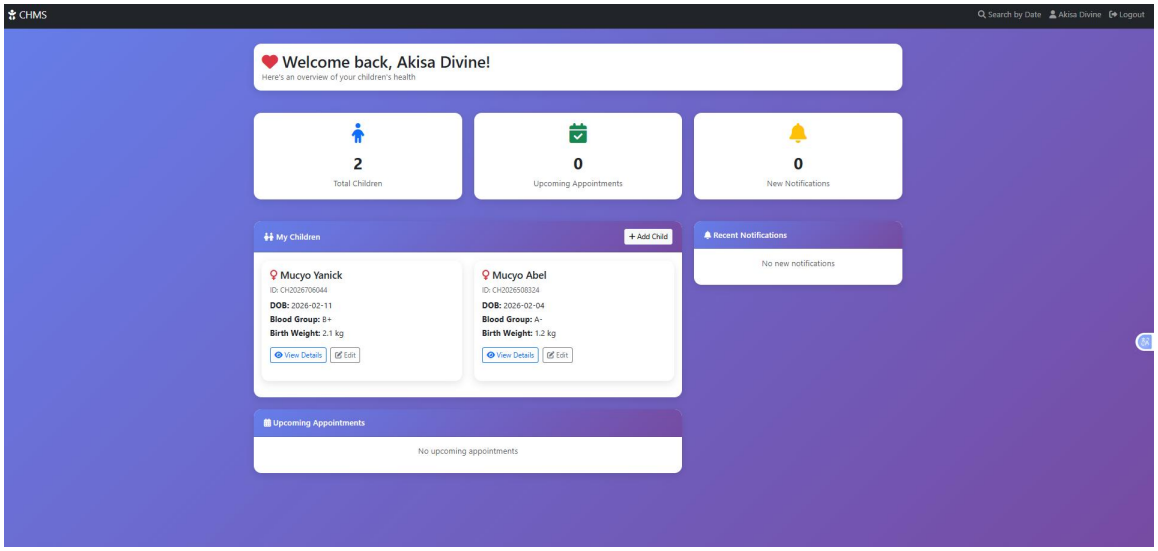
```
INSERT INTO children (unique_profile_id, full_name, date_of_birth, gender, birth_weight,
birth_height, blood_group, mother_id, father_name, father_phone, emergency_contact, address,
medical_history) VALUES
('CH2023001234', 'Emma Wilson', '2023-03-15', 'FEMALE', 3.2, 50.0, 'O+', 5, 'Robert Wilson',
'+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');
```

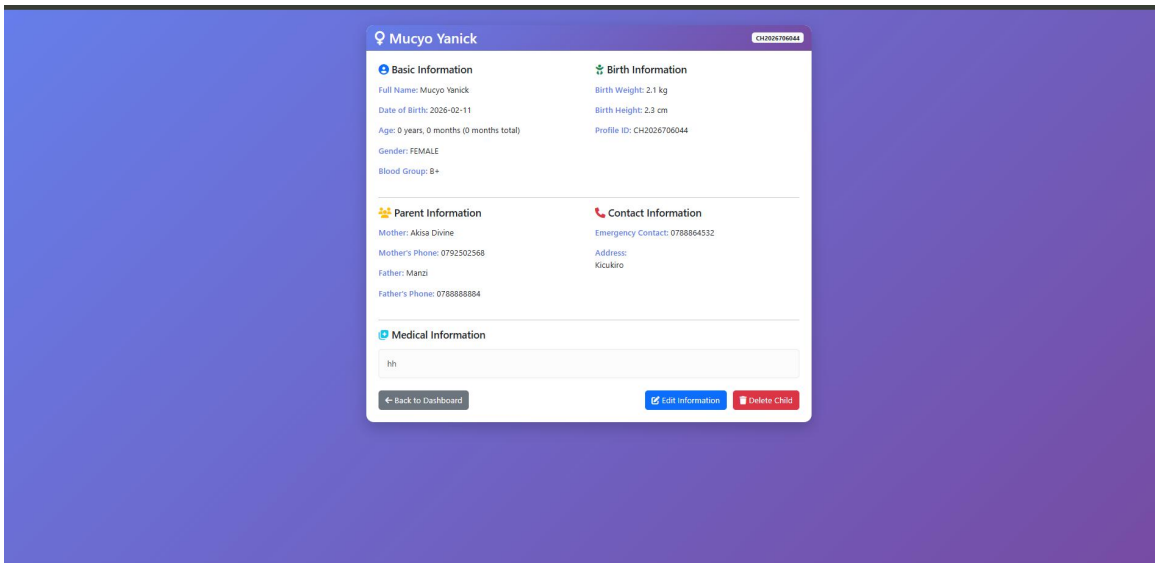


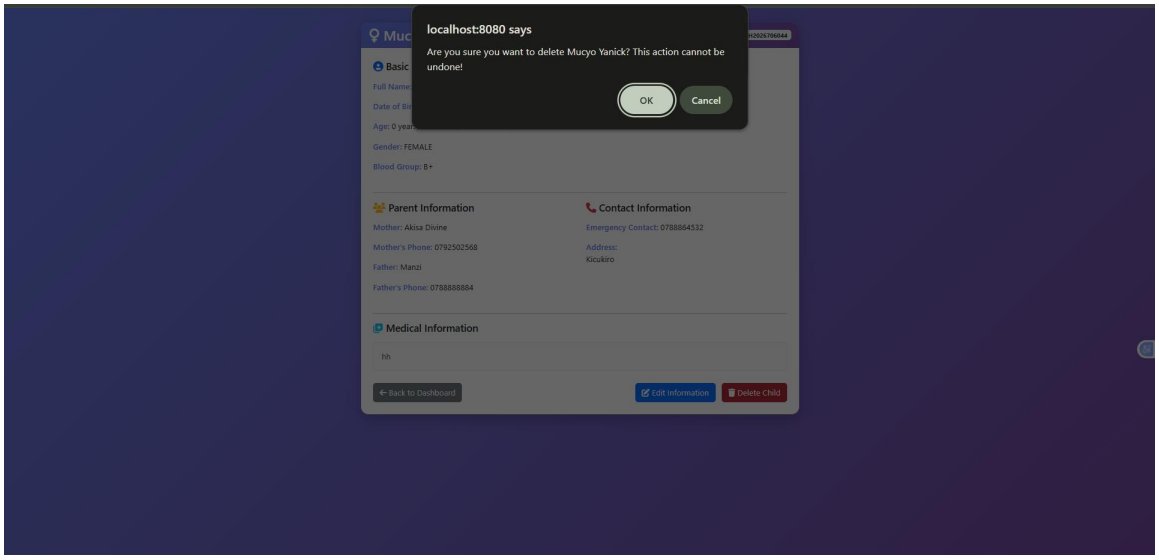
Database records before deletion(Mucyo Yannick)

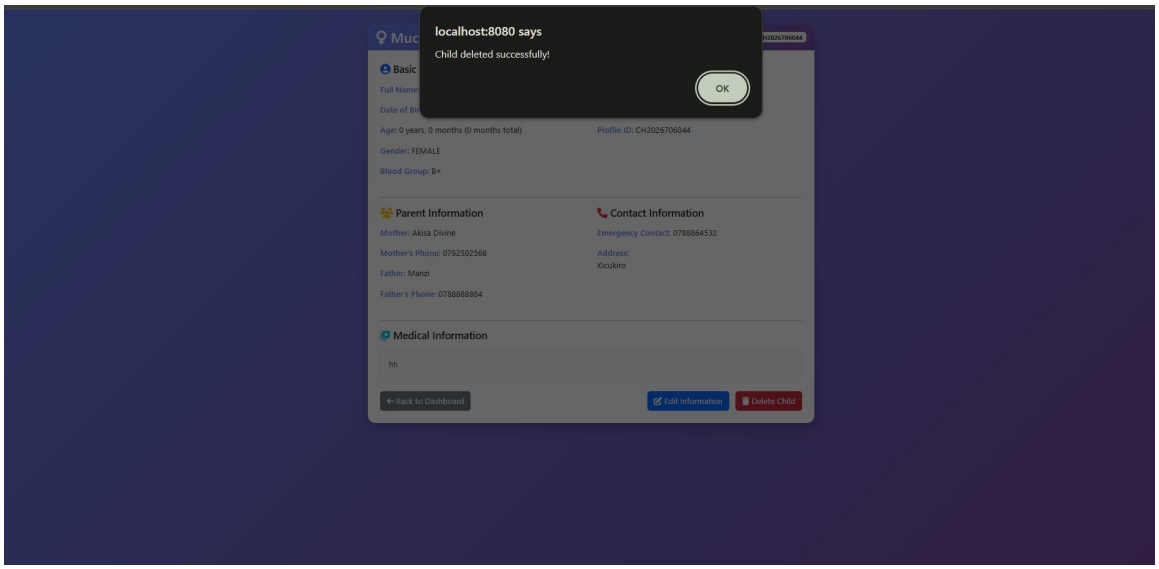


Before Delete Children

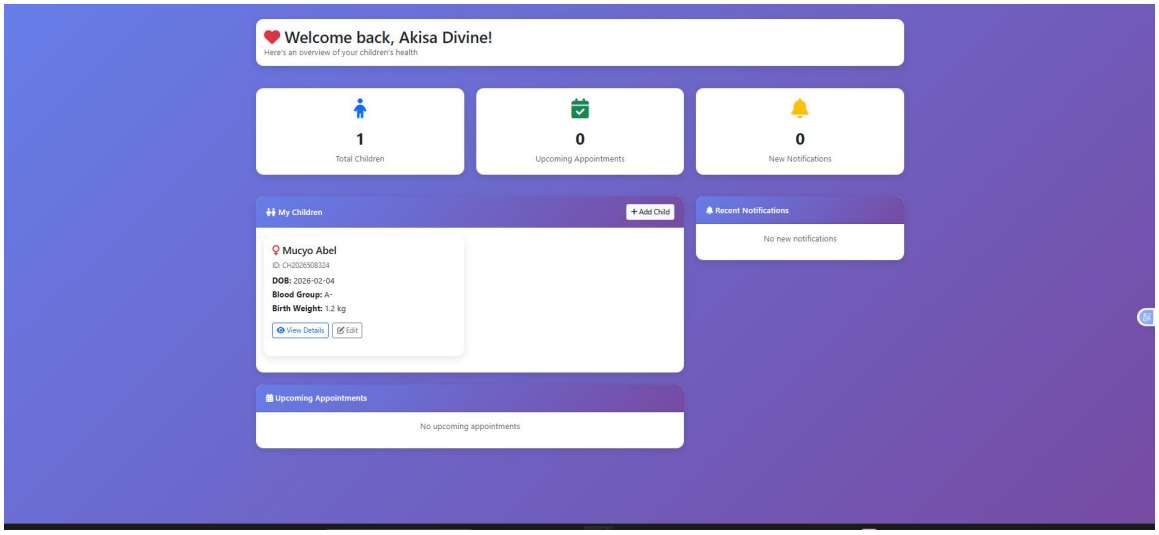



Mucyo Yanick
CH2026706044
Basic Information
Full Name: Mucyo Yanick
Date of Birth: 2026-02-11
Age: 0 years, 0 months (0 months total)
Gender: FEMALE
Blood Group: B+
Birth Information
Birth Weight: 2.1 kg
Birth Height: 2.3 cm
Profile ID: CH2026706044
Parent Information
Mother: Akisa Divine
Mother's Phone: 0792502568
Father: Manzi
Father's Phone: 0788888884
Contact Information
Emergency Contact: 0788864532
Address:
Kicukiro
Medical Information
hh
← Back to Dashboard
Edit Information
Delete Child


localhost:8080 says
Are you sure you want to delete Mucyo Yanick? This action cannot be undone!
OK
Cancel



After Delete Operation

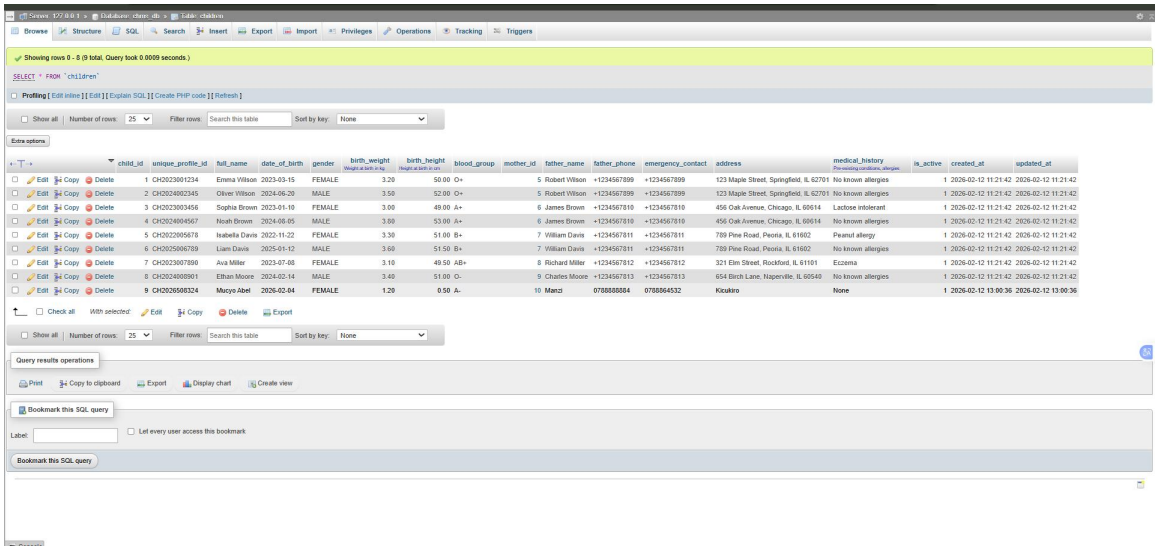


Before executing the DELETE operation, the mother had 2 children but now remain with one children

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 - 8 (9 total, Query took 0.0009 seconds.)

```
SELECT * FROM `children`
```

Profiling [Edit inline] [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 (Weight at birth in kg)	birth_height (Height at birth in cm)	blood_group	mother_id	father_name	father_phone	emergency_contact	address	medical_history (Pre-existing conditions, allergies)	is_active	created_at	updated_at
Edit Copy Delete	1	CH2023001234	Emma Wilson	2023-03-15	FEMALE	3.20	50.00	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
Edit Copy Delete	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	2026-02-12 11:21:42	2026-02-12 11:21:42
Edit Copy Delete	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
Edit Copy Delete	4	CH2024004567	Noah Brown	2024-06-05	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
Edit Copy Delete	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
Edit Copy Delete	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
Edit Copy Delete	7	CH2023007890	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
Edit Copy Delete	8	CH2024008901	Ethan Moore	2024-02-14	MALE	3.40	51.00	O-	9	Charles Moore	+1234567813	+1234567813	654 Birch Lane, Naperville, IL 60540	No known allergies	1	2026-02-12 11:21:42	2026-02-12 11:21:42
Edit Copy Delete	9	CH2026508324	Mucyo Abel	2026-02-04	FEMALE	1.20	0.50	A-	10	Manzi	0788888884	0788864532	Kicukiro	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

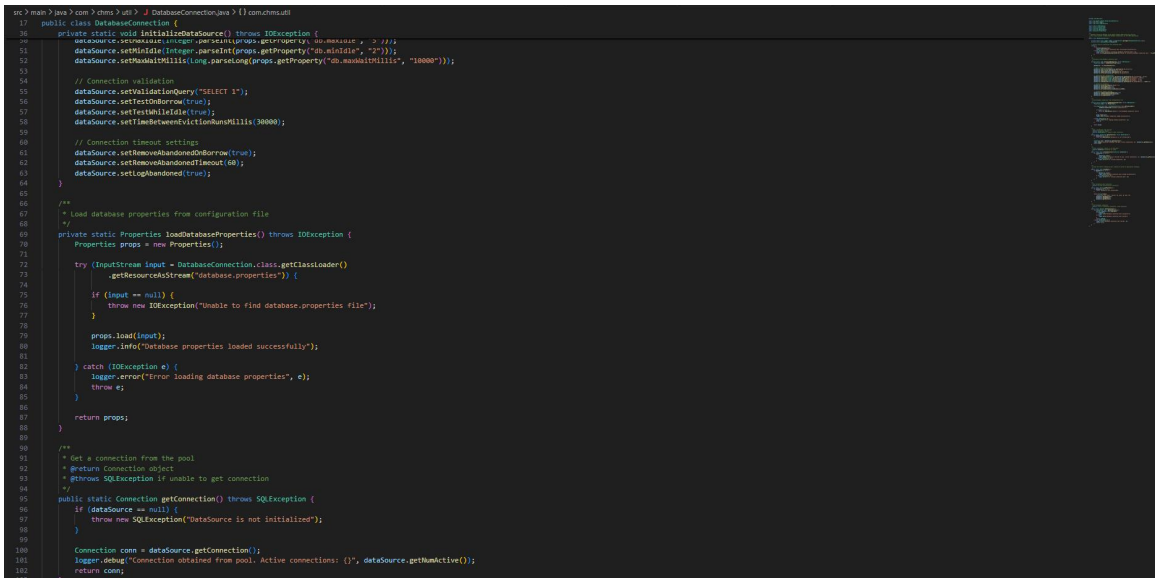
Bookmark this SQL query

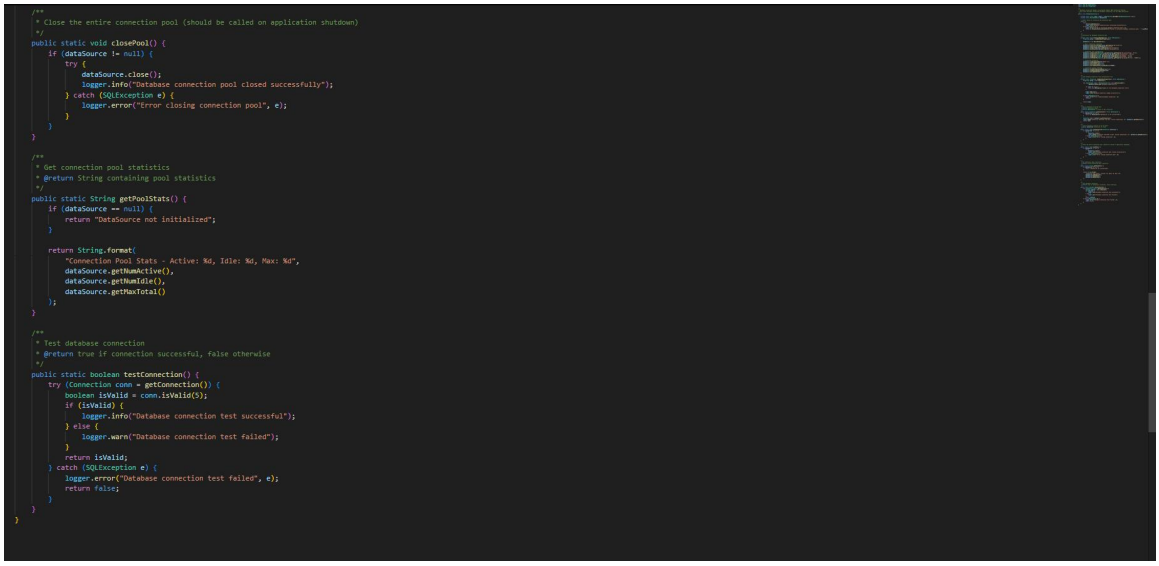
3. Java Implementation

3.1 Database Connection Code

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

```
src > main > java > com > chms > util > DatabaseConnection.java > {} com.chms.util
import java.util.Properties;

/**
 * Database Connection Manager using Apache Commons DBCP Connection Pooling
 * This class provides thread-safe database connections for the CHMS application
 */
public class DatabaseConnection {

    private static final Logger logger = LoggerFactory.getLogger(DatabaseConnection.class);
    private static BasicDataSource dataSource;

    // Static block to initialize the connection pool
    static {
        try {
            initializeDataSource();
            logger.info("Database connection pool initialized successfully");
        } catch (Exception e) {
            logger.error("Failed to initialize database connection pool", e);
            throw new ExceptionInInitializerError("Failed to initialize database connection pool: " + e.getMessage());
        }
    }

    /**
     * Initialize the database connection pool
     */
    private static void initializeDataSource() throws IOException {
        Properties props = loadDatabaseProperties();

        dataSource = new BasicDataSource();

        // Basic connection properties
        dataSource.setDriverClassName(props.getProperty("db.driver"));
        dataSource.setUrl(props.getProperty("db.url"));
        dataSource.setUsername(props.getProperty("db.username"));
        dataSource.setPassword(props.getProperty("db.password"));

        // Connection pool configuration
        dataSource.setInitialSize(Integer.parseInt(props.getProperty("db.initialSize", "5")));
        dataSource.setMaxTotal(Integer.parseInt(props.getProperty("db.maxTotal", "10")));
        dataSource.setMaxIdle(Integer.parseInt(props.getProperty("db.maxIdle", "5")));
        dataSource.setMinIdle(Integer.parseInt(props.getProperty("db.minIdle", "2")));
        dataSource.setMaxWaitMillis(Long.parseLong(props.getProperty("db.maxWaitMillis", "10000")));

        // Connection validation
        dataSource.setValidationQuery("SELECT 1");
        dataSource.setTestOnBorrow(true);
        dataSource.setTestWhileIdle(true);
        dataSource.setTimeBetweenEvictionRunsMillis(30000);

        // Connection timeout settings
        dataSource.setRemoveAbandonedOnBorrow(true);
        dataSource.setRemoveAbandonedTimeout(60);
        dataSource.setLogAbandoned(true);
    }
```





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.