```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

/// @title StudentDatabase
/// @notice Simple contract to store and manage student records (ID, name, grade)
/// @dev Demonstrates use of structs, arrays, mappings, receive/fallback, and basic CRUD
operations.
contract StudentDatabase {
    /// @dev Student record
    struct Student {
        uint256 studentId;
        string name;
        uint256 grade;
    }

    /// @notice Array of students (storage)
    Student[] private students;

    /// @dev Maps studentId => index in `students` array + 1. Zero means "not found".
    mapping(uint256 => uint256) private idToIndex;

    /// @notice Emitted when a new student is added
    event StudentAdded(uint256 indexed studentId, string name, uint256 grade);

    /// @notice Emitted when a student's grade or name is updated
    event StudentUpdated(uint256 indexed studentId, string name, uint256 grade);

    /// @notice Emitted when a student is removed
```

```
event StudentRemoved(uint256 indexed studentId);

/// @notice Emitted when the contract receives Ether
event Deposit(address indexed from, uint256 amount);

/// @notice Add a new student. Reverts if a student with the same ID already exists.
/// @param _studentId Unique identifier for the student
/// @param _name Student's name
/// @param _grade Student's grade
function addStudent(uint256 _studentId, string calldata _name, uint256 _grade) external {
   require(_studentId != 0, "studentId cannot be 0");
   require(idToIndex[_studentId] == 0, "studentId already exists");

   students.push(Student({ studentId: _studentId, name: _name, grade: _grade }));
   // store index+1 so that 0 means "not present"
   idToIndex[_studentId] = students.length;

   emit StudentAdded(_studentId, _name, _grade);
}

/// @notice Get the number of students stored
/// @return count Number of students
function getStudentCount() external view returns (uint256 count) {
   return students.length;
}

/// @notice Fetch a student by their ID
/// @param _studentId The student ID to lookup
```

```
    /// @return studentId The student's ID

    /// @return name The student's name

    /// @return grade The student's grade

    function getStudentById(uint256 _studentId)

        external

        view

        returns (uint256 studentId, string memory name, uint256 grade)

    {

        uint256 idx = idToIndex[_studentId];

        require(idx != 0, "student not found");

        Student storage s = students[idx - 1];

        return (s.studentId, s.name, s.grade);

    }


    /// @notice Update a student's name and/or grade

    /// @param _studentId The student ID to update

    /// @param _name New name (pass same name to keep unchanged)

    /// @param _grade New grade

    function updateStudent(uint256 _studentId, string calldata _name, uint256 _grade)
external {

        uint256 idx = idToIndex[_studentId];

        require(idx != 0, "student not found");

        Student storage s = students[idx - 1];

        s.name = _name;

        s.grade = _grade;


        emit StudentUpdated(_studentId, _name, _grade);

    }
```

```
/// @notice Remove a student by ID (swap-and-pop). Reverts if not found.
/// @param _studentId The student ID to remove
function removeStudent(uint256 _studentId) external {
    uint256 idx = idToIndex[_studentId];
    require(idx != 0, "student not found");

    uint256 removeIndex = idx - 1;
    uint256 lastIndex = students.length - 1;

    if (removeIndex != lastIndex) {
        // Move last student into the slot being removed
        Student storage lastStudent = students[lastIndex];
        students[removeIndex] = lastStudent;
        // Update mapping for moved student
        idToIndex[lastStudent.studentId] = removeIndex + 1;
    }

    // Remove last element
    students.pop();
    // Delete mapping entry
    delete idToIndex[_studentId];

    emit StudentRemoved(_studentId);
}

/// @notice Returns a student at a specific array index (0-based).
/// @dev Use only for enumeration; prefer getStudentById for by-ID fetches.
```

```
}

    /// @param _index Array index (0..count-1)
    /// @return studentId The student's ID
    /// @return name The student's name
    /// @return grade The student's grade
    function getStudentAtIndex(uint256 _index)
        external
        view
        returns (uint256 studentId, string memory name, uint256 grade)
    {
        require(_index < students.length, "index out of bounds");
        Student storage s = students[_index];
        return (s.studentId, s.name, s.grade);
    }

    /// @notice Receive function to accept plain Ether transfers
    receive() external payable {
        emit Deposit(msg.sender, msg.value);
    }

    /// @notice Fallback function to accept calls with data (and optional Ether)
    fallback() external payable {
        if (msg.value > 0) {
            emit Deposit(msg.sender, msg.value);
        }
    }
}
```