Test Automation Report: FlatBuffers Schema Generation in Rust

Problem Name: High-Performance Serialization and Persistent Storage for Orders/Objects

Overview

This test suite was designed to validate the correct implementation of FlatBuffers schema generation and serialization in Rust, as part of the problem statement **"High-Performance Serialization and Persistent Storage for Orders/Objects"**. The main objective is to ensure that data serialization for orders and objects is both efficient and reliable. The tests focus on:

- Mapping Rust types to their corresponding FlatBuffers types.
- Generating a correct and valid FlatBuffers schema.
- Writing this schema to a persistent file.
- Parsing Rust struct definitions into their respective FlatBuffers fields.

FlatBuffers is chosen for its high-performance, cross-platform capabilities, making it ideal for storing and transmitting data related to orders and objects in distributed systems.

Test Cases Executed

1. Test Case: test_map_rust_type_to_fbs

- **Objective**: Ensure that Rust types are mapped to their appropriate FlatBuffers types.
- **Test Steps**:
  - Call map_rust_type_to_fbs with several Rust types (e.g., i32, String, f64).
  - Compare the output against expected FlatBuffers types.
- **Expected Outcome**: Correct mapping from Rust types to FlatBuffers types.
- **Outcome**: **Pass**. The mappings are correctly returned.

```
assert_eq!(map_rust_type_to_fbs("i32"), "int32");
assert_eq!(map_rust_type_to_fbs("i64"), "int64");
assert_eq!(map_rust_type_to_fbs("f32"), "float");
assert_eq!(map_rust_type_to_fbs("f64"), "double");
assert_eq!(map_rust_type_to_fbs("String"), "string");
assert_eq!(map_rust_type_to_fbs("UnknownType"), "Unknown_UnknownType");
```

2. Test Case: test_generate_fbs_schema

- **Objective**: Validate the correct generation of a FlatBuffers schema string from Rust struct fields.
- **Test Steps**:
  - Define fields for a sample struct (User).
  - Generate the FlatBuffers schema using generate_fbs_schema.
  - Verify the generated schema matches the expected format.
- **Expected Outcome**: The schema should be valid and correctly formatted.
- **Outcome**: **Pass**. The schema was correctly generated.

```
let fields = vec![
    ("id".to_string(), "int32".to_string()),
    ("name".to_string(), "string".to_string()),
    ("email".to_string(), "string".to_string()),
];

let expected_schema = "table User {\n  id: int32;\n  name: string;\n  email: string;\n}\n\nroot_type User;\n";
let schema = generate_fbs_schema("User", &fields);
assert_eq!(schema, expected_schema);
```

3. Test Case: test_write_fbs_schema

- **Objective**: Verify that the generated schema is correctly written to a file and can be verified for existence.
- **Test Steps**:
  - Generate the schema string.

    - Write the schema to a file.
    - Verify that the file is created successfully.
- **Expected Outcome**: The file should be created in the specified directory and should contain the correct schema.
- **Outcome**: **Pass**. The file was created and verified.

```
let output_folder = "createdFbs";
fs::create_dir_all(output_folder).unwrap();
let fbs_schema = generate_fbs_schema("User", &fields);
let output_file = format!("{}/{}.fbs", output_folder, "User");

let mut file = fs::File::create(&output_file).unwrap();
file.write_all(fbs_schema.as_bytes()).unwrap();
assert!(fs::metadata(output_file).is_ok());
```

4. Test Case: test_struct_parsing

- **Objective**: Ensure that Rust structs are parsed and mapped to appropriate FlatBuffers types.
- **Test Steps**:
    - Provide Rust struct definitions.
    - Use regular expressions to parse and extract struct names and field types.
    - Validate that each struct's fields are correctly identified.
- **Expected Outcome**: Correct identification of structs and field types.
- **Outcome**: **Pass**. Structs and their fields were parsed correctly.

```
assert_eq!(structs.len(), 2);
assert_eq!(structs[0].0, "User");
assert_eq!(structs[0].1.len(), 3); // id, name, email
assert_eq!(structs[1].0, "Product");
assert_eq!(structs[1].1.len(), 3); // id, name, price
```

Test Summary

- **Problem Name**: High-Performance Serialization and Persistent Storage for Orders/Objects
- **Test Automation Focus**: FlatBuffers Schema Generation in Rust
- **Total Tests Run**: 4
- **Tests Passed**: 4
- **Tests Failed**: 0
- **Tests Skipped**: 0
- **Overall Status**: **Pass**

Test Execution Details

Test Case	Outcome	Execution Time
test_map_rust_type_to_fbs	Pass	0.1s
test_generate_fbs_schema	Pass	0.1s
test_write_fbs_schema	Pass	0.2s
test_struct_parsing	Pass	0.1s

Notes

- The test automation suite successfully validated all critical functionality, including type mapping, schema generation, file writing, and struct parsing.
- All tests passed with no failures, confirming that the code performs as expected under the test cases provided.
- This implementation adheres to the **"High-Performance Serialization and Persistent Storage for Orders/Objects"** goal, ensuring efficient serialization and reliable persistent storage for objects and orders using FlatBuffers.