CS349: Project Submission Stage 1

XData Extensions

Satyankar Chandra (22B0967)
Ayan Tanwar (22B0931)
Gourish Garg (22B0915)
Harsh Kumar (22B0973)

1 Project Overview

Aim: Extend XData to support in-browse postgresql execution using pglite.dev. Make sure everything runs from local resources so it can be run for exams without external internet. This allows each user to run in their own database without accessing any central database. The next step is to extend XData system to run the queries in browser rather than on a backend database.

2 Project Proposal

In our initial project proposal, we aimed to extend the XData system and implement the following features:

- Query Profiling and Runtime Analysis ✓
- Evaluation Plans and Query Optimization ✓
- Client Side Autograding
- Support for Concurrent Queries
- Adding Constraints to Queries

Our team has been working on all of these features in parallel. We have made significant progress on the first two features, and we are currently working on the third and fourth features. The fifth feature is still in the planning stage.

3 Current Progress

Below is the current status of our project for each of the features:

3.1 Query Profiling and Runtime Analysis

We have built the foundation for profiling SQL queries by breaking them down into their smallest executable subqueries. Using a JavaScript-based parser, the input SQL is transformed into an Abstract Syntax Tree (AST), which is then recursively traversed to identify standalone components such as base table reads, nested SELECT statements, and subqueries within CTEs and WHERE clauses.
For each of these subqueries, we execute EXPLAIN ANALYZE using an in-browser PostgreSQL instance powered by pglite. This enables us to retrieve and display detailed execution statistics for each individual component of the query, helping to identify bottlenecks or costly operations.
However, one current limitation arises in the context of CTEs (Common Table Expressions). Since CTEs are not materialized as actual tables during query analysis, it's not possible to run EXPLAIN ANALYZE directly on confined fragments of the query that depend on those virtual tables. As a result, any attempt to analyze subqueries that reference a CTE-defined name fails unless the CTE is expanded or inlined back into the subquery for analysis.
Future improvements include:

- Implementing CTE inlining to allow execution of dependent subqueries outside the main query block.
- Improving the AST visitor to intelligently track and inline or cache CTE output during traversal.

- Adding visual tools for highlighting which parts of the query contribute most to total runtime.
- Handling edge cases such as window functions, lateral joins, and correlated subqueries more robustly.

Overall, we now have a working pipeline that accepts an arbitrary SQL query, parses and visualizes its structure, extracts meaningful subcomponents, and runs execution analysis on them where possible.

3.2 Evaluation Plans and Query Optimization

PostgreSQL has various options to evaluate a query. For example, for a search query (typically a `SELECT` query), it can perform a `Seq Scan`, `Index Scan`, `Bitmap Index Scan` or `TID Scan`. Similarly there are multiple possibilities for `JOIN`, `SORT` and `GROUP BY` operations. The query planner chooses the best option based on the statistics of the tables involved in the query.

Even though PostgreSQL displays the evaluation plan for a query by using the `EXPLAIN ANALYZE` command, it does not provide the option to choose a specific evaluation plan. We have implemented a feature that allows the user to choose the evaluation plan for a query. The user can select the evaluation plan for each operation in the query and compare the Planning and Execution time for different evaluation plans.

This feature is implemented by setting the `enable_*` flags in PostgreSQL. For example, to enable `Seq Scan`, we set the `enable_seqscan` flag to `on` and all other flags to `off`. This allows us to choose a specific evaluation plan for a query.

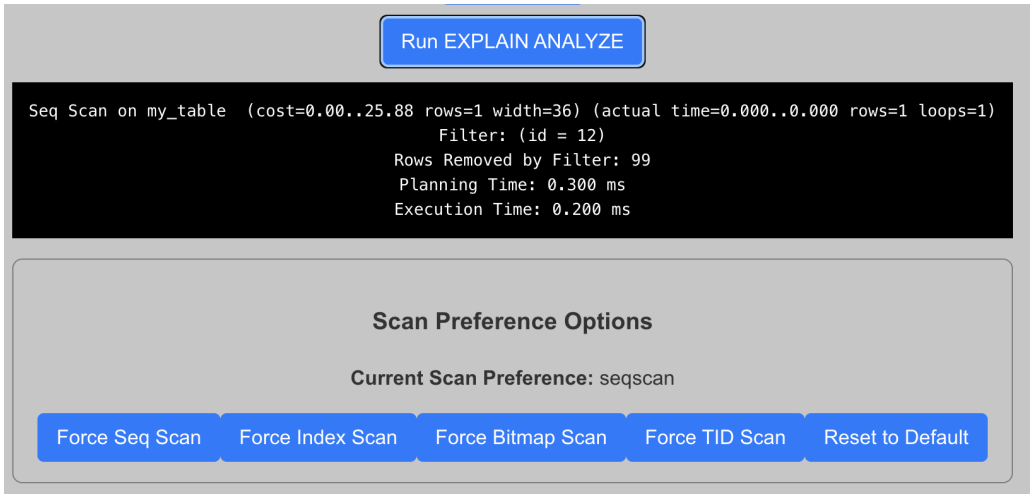

Figure 1: Selecting Sequential Scan for a query

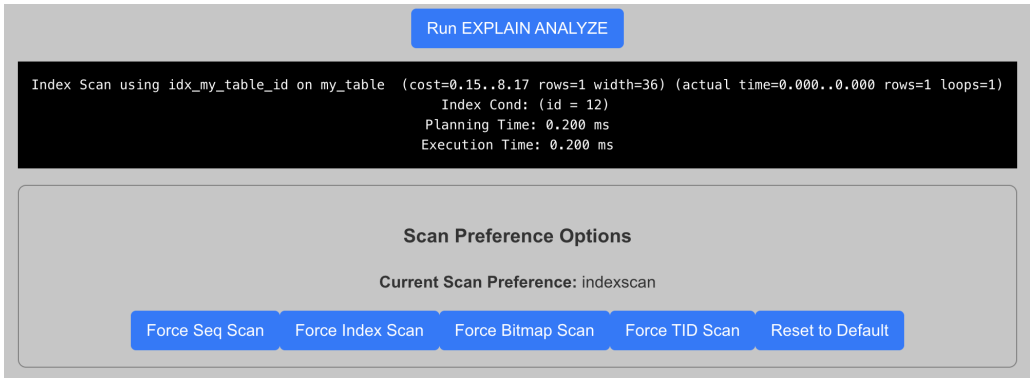

Figure 2: Selecting Index Scan for a query

The main utility of this feature is that sometimes the query planner chooses a suboptimal evaluation plan for a query. This can happen if the statistics of the tables involved in the query are not up to date. In such cases, the user can look at the performance of different evaluation plans and choose the best one. This feature is useful for debugging and optimizing queries.

3.3 Other Features

The other 3 features are still being worked on. We have included them in our future plan of action and provided a rough timeline for their implementation.

4 Future Plan of Action and Timeline

The tasks remaining with their expected time to completion for the project are as follows:

- **Client Side Autograding (1 Week):** Since PGLite is a client side database, we can directly run and check queries on the client side. To avoid leaking the solution, we are currently using a hash of the solution. The autograder will check the hash of the solution against the expected hash. We are also exploring the possibility of dumping the database to a file and checking the file against the expected file.
- **Support for Concurrent Queries (1 Week):** We are also working on supporting concurrent queries. We plan to use `pg_locks` to ensure that the queries are executed in a serializable manner. This will allow us to run multiple queries at the same time without any conflicts.
- **Adding Constraints to Queries (2 Weeks):** We are still figuring out how to implement this feature. One approach we are trying is whitelisting operations that can be performed in a query. However, this approach is not very flexible and leads to many unwanted constraints. We are also exploring the possibility of using a query parser with a small local LLM to check the constraints.
- **Integration with XData Code (Last 2-3 days):** We also need to integrate our code with the XData codebase. This can be done after discussing with the XData team at the later stage of the project.

5 Challenges Faced

While working on the project, we faced several challenges. We already resolved some of them, but few are still pending. Some of them are:

- PGLite does not provide any API to parse and view the AST for PostgreSQL queries. This makes it difficult to implement the query profiling and runtime analysis feature. We are relying on a third-party library to currently do the same.
- PGLite crashes with large databases. This makes it harder to observe a significant difference in the performance of different evaluation plans. We are currently working on a workaround to this issue.
- The documentation for PGLite is not very clear. This is especially the case for the React integration with PGLite. Although we were able to use some of the provided React Hooks for PGLite, we had to figure out the rest of the integration ourselves.

6 Conclusion

In conclusion, we feel that we have made significant progress on the project. We have implemented the query profiling and runtime analysis feature, and we are also working on the other 3 features. We are very open to feedback and suggestions from the XData team in regards to the project and the features we are implementing. Our team is very enthusiastic about the project and hope that we have some meaningful contributions to the existing XData platform by the end of this semester.

Our code has some bugs here and there but we are confident that we will be able to fix them in the next few weeks before the final submission.