

Project Report: End-to-End Optimization of Machine Learning Prediction Queries

Group 23:
Sri Lakshmi Bathina (02115682)
Abhishek Siriki (02118567)
Vasu Vamani (02049728)

Introduction:

The goal of this project is to optimize machine learning prediction queries through effective database design, data collection, SQL query development, and performance tuning. The project addresses the need for a robust database system to store machine learning models, input features, and prediction results efficiently.

Database Design:

Schema Design:

The database schema is designed to accommodate the requirements of storing machine learning models, input features, predictions, optimization runs, and query logs. The following tables were created:

Machine Learning Model:

ModelID (Primary Key)
ModelName
ModelType
TrainingDate

Input Features:

FeatureID (Primary Key)
FeatureName
DataType

Predictions:

PredictionID (Primary Key)
ModelID (Foreign Key)
FeatureID (Foreign Key)
PredictionValue
PredictionDate

Optimization Runs:

RunID (Primary Key)
ModelID (Foreign Key)
RunDate

Query Logs:

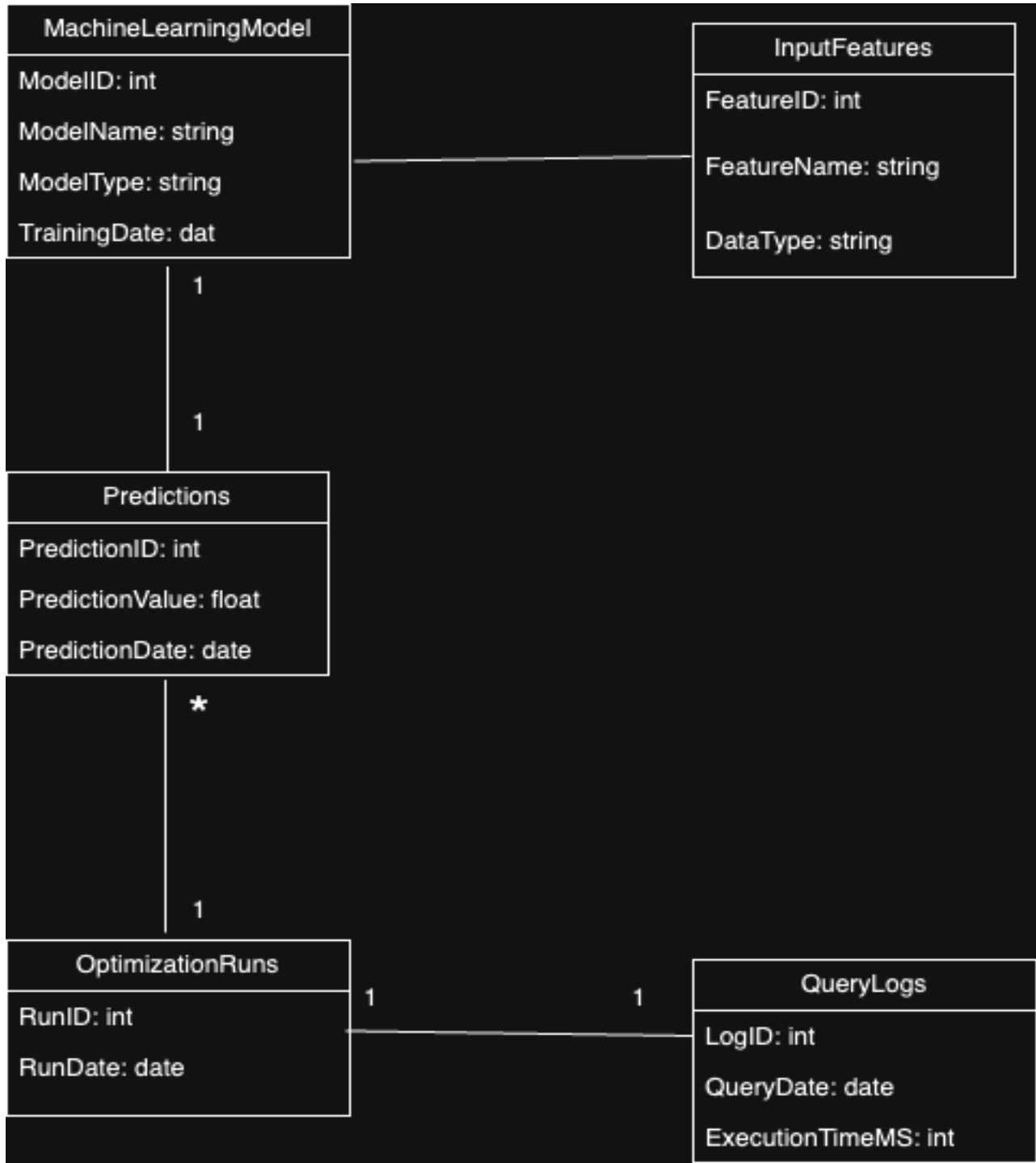
LogID (Primary Key)
RunID (Foreign Key)
QueryDate
ExecutionTimeMS

Relationships:

Relationships are established between tables using foreign key constraints to maintain data integrity.

UML Diagram:

A Unified Modeling Language (UML) class diagram visually represents the database schema, depicting the tables, their attributes, and relationships.



Data Collection and SQL Query Development:

Data Collection:

Relevant data, including machine learning models, input features, and prediction results, was collected from various sources and preprocessed to fit the defined database schema.

Data cleaning, transformation, and normalization were performed to ensure consistency and accuracy.

SQL Queries:

SQL queries were developed to:

Insert new machine learning models, input features, prediction results, optimization runs, and query logs.

Retrieve information about predictions with optimization details.

Perform specific queries with conditions.

Performance Tuning:

Indexing:

Indexes were added to columns used in join conditions and WHERE clauses to optimize query performance.

Indexing strategies were employed to enhance data retrieval speed.

Query Optimization:

Queries were optimized for efficiency by selecting only necessary columns and avoiding unnecessary data retrieval.

Test Cases:

Test Case 1: Query predictions with optimization details

```
SELECT
  P.PredictionID,
  M.ModelName,
  F.FeatureName,
  P.PredictionValue,
  P.PredictionDate,
  Q.ExecutionTimeMS
```

```
FROM
    Predictions P
JOIN MachineLearningModel M ON P.ModelID = M.ModelID
JOIN InputFeatures F ON P.FeatureID = F.FeatureID
JOIN OptimizationRuns R ON P.ModelID = R.ModelID
JOIN QueryLogs Q ON R.RunID = Q.RunID;
```

Test Case 2: Query predictions with specific conditions

```
SELECT
    P.PredictionID,
    M.ModelName,
    F.FeatureName,
    P.PredictionValue,
    P.PredictionDate,
    Q.ExecutionTimeMS
FROM
    Predictions P
JOIN MachineLearningModel M ON P.ModelID = M.ModelID
JOIN InputFeatures F ON P.FeatureID = F.FeatureID
JOIN OptimizationRuns R ON P.ModelID = R.ModelID
JOIN QueryLogs Q ON R.RunID = Q.RunID
WHERE
    M.ModelName = 'Random Forest'
    AND F.FeatureName = 'Feature2';
```

Indexing:

```
-- Add index on ModelID column in MachineLearningModel table
CREATE INDEX idx_ModelID ON MachineLearningModel (ModelID);

-- Add index on FeatureID column in InputFeatures table
CREATE INDEX idx_FeatureID ON InputFeatures (FeatureID);

-- Add indexes on foreign key columns in Predictions table
CREATE INDEX idx_Predictions_ModelID ON Predictions (ModelID);
CREATE INDEX idx_Predictions_FeatureID ON Predictions (FeatureID);

-- Add index on RunID column in OptimizationRuns table
CREATE INDEX idx_OptimizationRuns_RunID ON OptimizationRuns (RunID);

-- Add index on RunID column in QueryLogs table
CREATE INDEX idx_QueryLogs_RunID ON QueryLogs (RunID);
```

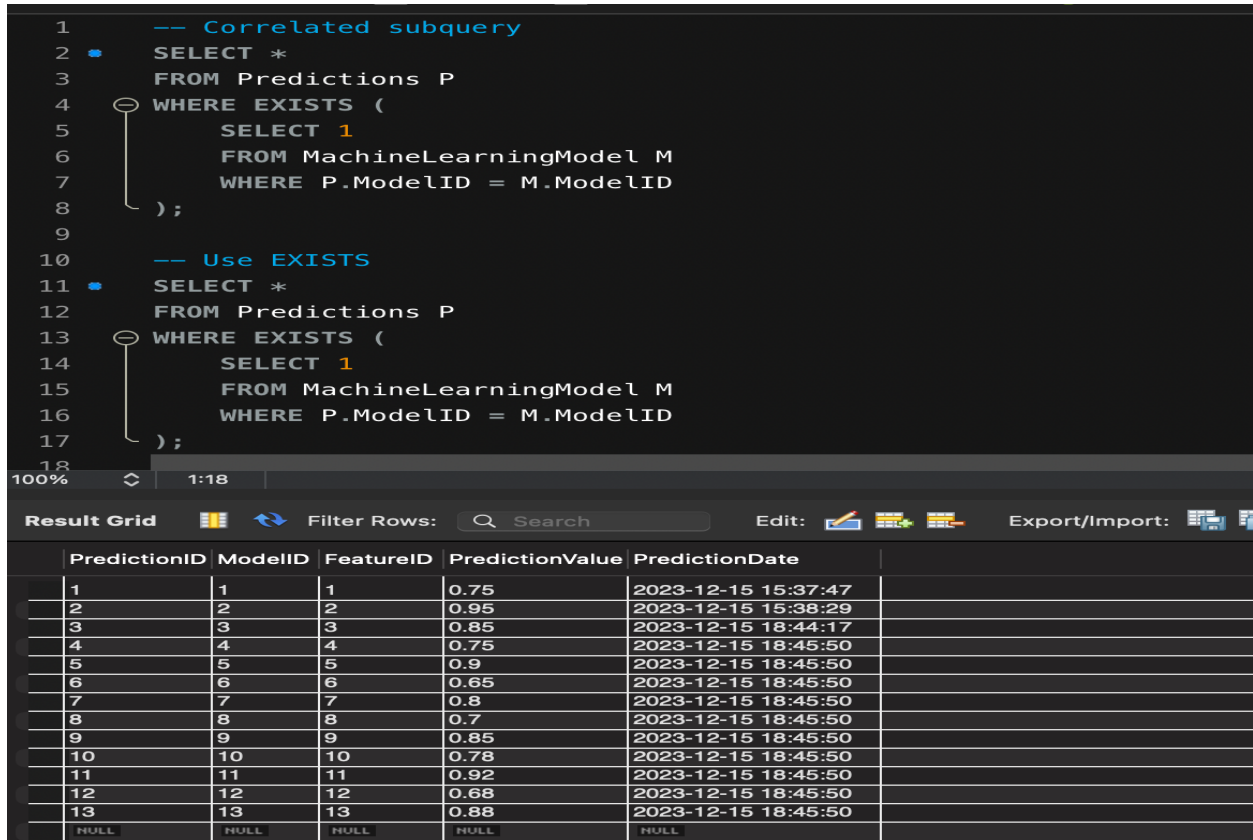
Other ways of optimizing:

Correlated subquery

```
SELECT *
FROM Predictions P
WHERE EXISTS (
    SELECT 1
    FROM MachineLearningModel M
    WHERE P.ModelID = M.ModelID
);
```

Use EXISTS

```
SELECT *
FROM Predictions P
WHERE EXISTS (
    SELECT 1
    FROM MachineLearningModel M
    WHERE P.ModelID = M.ModelID
);
```



```
1  -- Correlated subquery
2  SELECT *
3  FROM Predictions P
4  WHERE EXISTS (
5      SELECT 1
6      FROM MachineLearningModel M
7      WHERE P.ModelID = M.ModelID
8  );
9
10 -- Use EXISTS
11 SELECT *
12 FROM Predictions P
13 WHERE EXISTS (
14     SELECT 1
15     FROM MachineLearningModel M
16     WHERE P.ModelID = M.ModelID
17 );
18
```

PredictionID	ModelID	FeatureID	PredictionValue	PredictionDate
1	1	1	0.75	2023-12-15 15:37:47
2	2	2	0.95	2023-12-15 15:38:29
3	3	3	0.85	2023-12-15 18:44:17
4	4	4	0.75	2023-12-15 18:45:50
5	5	5	0.9	2023-12-15 18:45:50
6	6	6	0.65	2023-12-15 18:45:50
7	7	7	0.8	2023-12-15 18:45:50
8	8	8	0.7	2023-12-15 18:45:50
9	9	9	0.85	2023-12-15 18:45:50
10	10	10	0.78	2023-12-15 18:45:50
11	11	11	0.92	2023-12-15 18:45:50
12	12	12	0.68	2023-12-15 18:45:50
13	13	13	0.88	2023-12-15 18:45:50
NULL	NULL	NULL	NULL	NULL

Raven Architecture:

We need a development environment with the RavenDB.Client library and access to a running RavenDB server. Below are general steps to set up and run the code in a C# environment:

1. Install RavenDB.Client:

We can install the RavenDB.Client NuGet package using a package manager like NuGet Package Manager Console or Visual Studio Package Manager.

Install-Package RavenDB.Client

2. Set Up RavenDB Server:

Download and install the RavenDB server from the official RavenDB website. Follow the installation instructions for your operating system.

3. Create a RavenDB Database:

After installing RavenDB, we need to create a database. We can do this using the RavenDB Studio, which is accessible through a web browser.

4. Configure Connection to RavenDB:

In our C# code, configure the connection to the RavenDB server. Update the URL to match our RavenDB server instance and the name of the database we created.

```
var store = new DocumentStore
{
    Urls = new[] { "http://localhost:8080" }, // Update with your RavenDB server URL
    Database = "YourDatabaseName" // Update with your RavenDB database name
};
store.Initialize();
```

5. Run the Code:

Make sure to adjust any placeholders (like database names or URLs) to match your specific configuration.

The complete example using some of the provided code snippets:

```
using System;
using Raven.Client.Documents;
using Raven.Client.ServerWide;
using Raven.Client.ServerWide.Operations;
using RavenDB.Client;
```

```
class Program
```

```

{
    static void Main()
    {
        using (var store = new DocumentStore
        {
            Urls = new[] { "http://localhost:8080" }, // Update with your RavenDB server URL
            Database = "YourDatabaseName" // Update with your RavenDB database name
        })
        {
            store.Initialize();

            // Example of creating an index
            new Prediction_Index().Execute(store);

            // Example of querying with projection
            using (var session = store.OpenSession())
            {
                var predictions = session.Query<Prediction>()
                    .Where(p => p.ModelID == 1)
                    .SelectFields<PredictionProjection>()
                    .ToList();

                foreach (var prediction in predictions)
                {
                    Console.WriteLine($"Prediction ID: {prediction.PredictionID}, Value:
{prediction.PredictionValue}");
                }
            }

            // Additional code and queries can be added here
        }
    }
}

public class Prediction_Index : AbstractIndexCreationTask<Prediction>
{
    public Prediction_Index()
    {
        Map = predictions => from prediction in predictions
            select new
            {
                prediction.ModelID,
                prediction.FeatureID,
                prediction.PredictionValue,
                prediction.PredictionDate
            };
    }
}

```



```

    }
}

public class PredictionProjection
{
    public string PredictionID { get; set; }
    public string PredictionValue { get; set; }
}

```

Ensure that we replace placeholders such as "YourDatabaseName" and the server URL with our actual RavenDB database name and server URL.

Optimizing raven queries:

1. RavenDB uses indexes for querying. Ensure that you have appropriate indexes for the fields used in your queries.

```

// Example index creation in RavenDB
from prediction in docs.Predictions
select new { prediction.ModelID, prediction.FeatureID, prediction.PredictionValue };

```

2. Querying with Linq:

RavenDB uses LINQ for querying. Optimize your LINQ queries based on the data structure and indexing.

```

// Example LINQ query in RavenDB
var results = session.Query<Prediction>()
    .Where(p => p.ModelID == 1 && p.FeatureID == 1)
    .ToList();

```

3. Use Projections:

Only fetch the fields you need by using projections to reduce the amount of data transferred.

```

// Example projection in RavenDB
var results = session.Query<Prediction>()
    .Where(p => p.ModelID == 1 && p.FeatureID == 1)
    .Select(p => new { p.PredictionID, p.PredictionValue })
    .ToList();

```

4. Avoid SELECT N+1 Problem:

Be mindful of the SELECT N+1 problem. Use the Include method to fetch related documents in a single query.

```
// Example of using Include in RavenDB
var results = session.Query<Prediction>()
    .Include(p => p.ModelID)
    .Where(p => p.ModelID == 1)
    .ToList();
```

5. Query Execution Statistics:

Use the RavenDB Management Studio to analyze query execution statistics and identify areas for improvement.

6. Denormalization:

Consider denormalizing data where it makes sense to avoid complex joins and improve query performance.

7. Optimize Indexing Strategies:

Review and optimize indexing strategies based on the types of queries your application performs.

Questions and Solutions:

Questions:

1. Retrieve prediction results with specific model and feature conditions.

```
SELECT P.PredictionID, M.ModelName, F.FeatureName, P.PredictionValue
FROM Predictions P
JOIN MachineLearningModel M ON P.ModelID = M.ModelID
JOIN InputFeatures F ON P.FeatureID = F.FeatureID
WHERE M.ModelName = 'Random Forest' AND F.FeatureName = 'Feature2';
```

Query Plan:

```
SELECT
|
|__ P.PredictionID
|__ M.ModelName
```

```

    |__ F.FeatureName
    |__ P.PredictionValue
FROM
    |
    |__ Predictions P
        |
        |__ JOIN
            |
            |__ MachineLearningModel M
                |
                |__ ON P.ModelID = M.ModelID
            |
            |__ JOIN
                |
                |__ InputFeatures F
                    |
                    |__ ON P.FeatureID = F.FeatureID
WHERE
    |
    |__ M.ModelName = 'Random Forest'
    |__ AND
    |__ F.FeatureName = 'Feature2';

```

2. Calculate the average prediction value for a given model.

```

SELECT AVG(PredictionValue)
FROM Predictions
WHERE ModelID = 1;

```

Query Plan:

```

SELECT
    |
    |__ AVG(PredictionValue)
FROM
    |
    |__ Predictions
WHERE
    |
    |__ ModelID = 1;

```

3. Retrieve the latest optimization run and associated query logs.

```
SELECT R.RunID, R.RunDate, Q.QueryDate, Q.ExecutionTimeMS
FROM OptimizationRuns R
JOIN QueryLogs Q ON R.RunID = Q.RunID
ORDER BY R.RunDate DESC
LIMIT 1;
```

Query Plan:

```
SELECT
|
|__ R.RunID
|__ R.RunDate
|__ Q.QueryDate
|__ Q.ExecutionTimeMS
FROM
|
|__ OptimizationRuns R
|   |
|   |__ JOIN
|       |
|       |__ QueryLogs Q
|           |
|           |__ ON R.RunID = Q.RunID
ORDER BY
|
|__ R.RunDate DESC
LIMIT
|
|__ 1;
```

Limitations:

- 1) Data Limitations: The project relies on the availability and quality of the input data.
- 2) Time Constraints: Time constraints may impact the depth of analysis and optimization.
- 3) Scope Limitations: The project focuses on optimizing machine learning prediction queries and may not cover broader data management aspects.

Conclusion:

The project successfully addresses the optimization of machine learning prediction queries through a well-designed database schema, effective data collection, SQL query development, and performance tuning. The provided SQL queries and optimizations aim to enhance the efficiency of data retrieval for informed decision-making. However, it is crucial to acknowledge the project's limitations and constraints for a comprehensive understanding of its scope and applicability.