Optimizing Access Across Multiple Hierarchies in Data Warehouses Query Rewriting

Student: Jade Cheng

Date: May 02, 2011

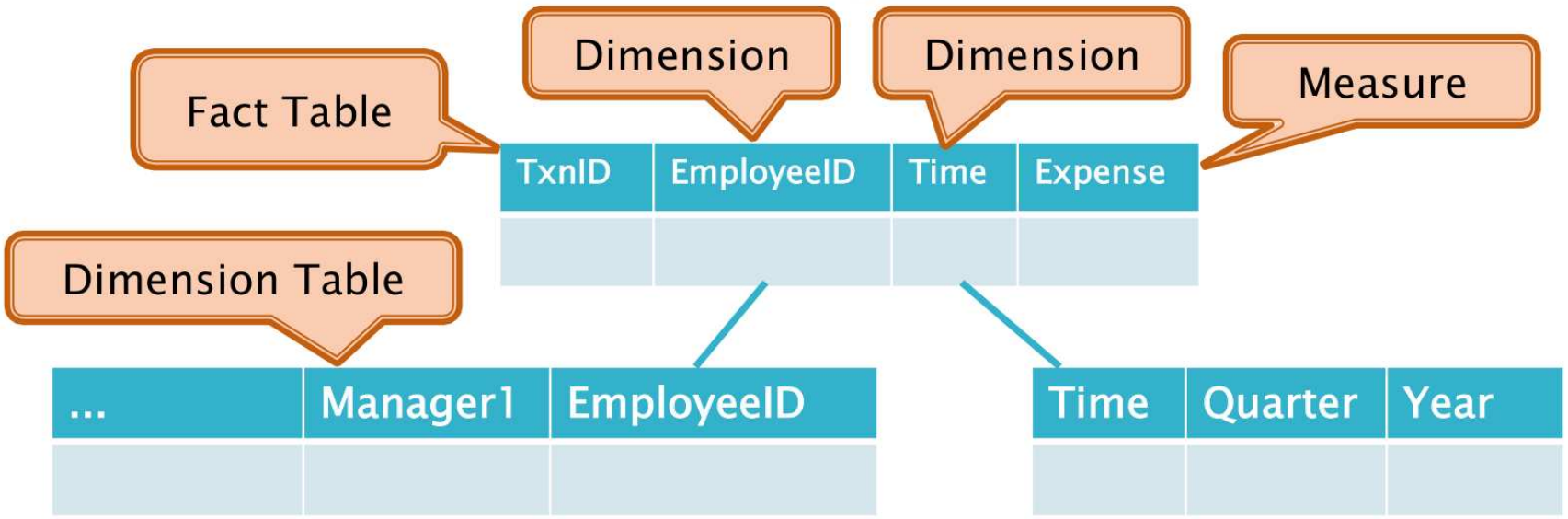
Supervisor: Dr. Lipyeow Lim, PhD

Introduction

Data Warehousing

OLAP data mart consists of a fact table and several dimensions.

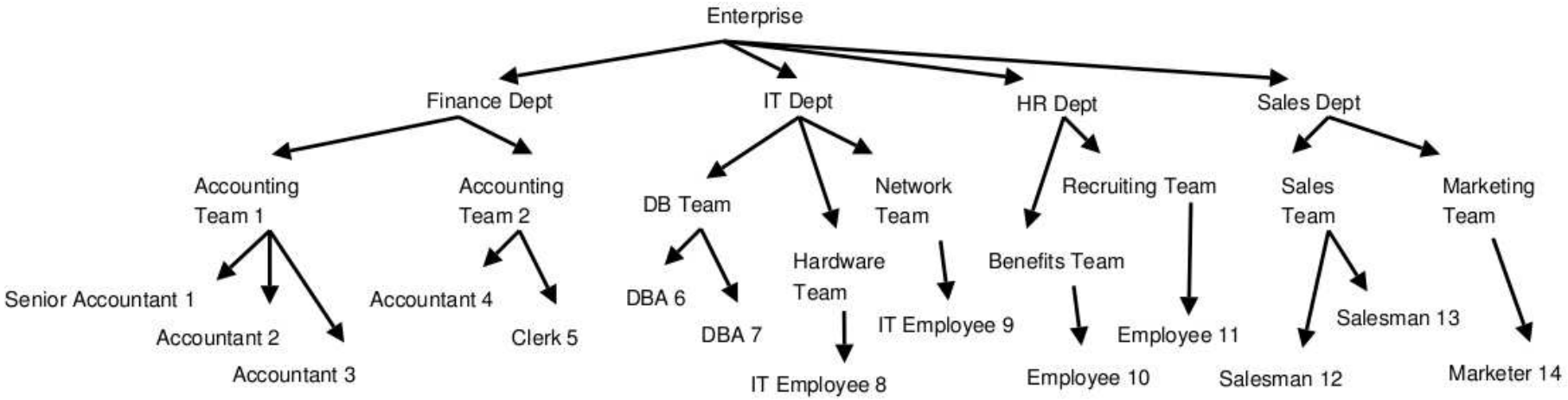
Each dimension is associated with multiple complex and unbalanced hierarchies.



Introduction

Primary Hierarchies

A set of primary hierarchies are defined on each common fact table.

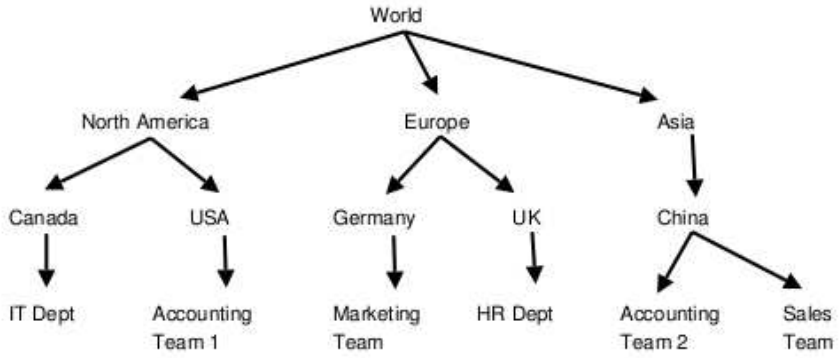


(a) Primary hierarchy for the organization dimension.

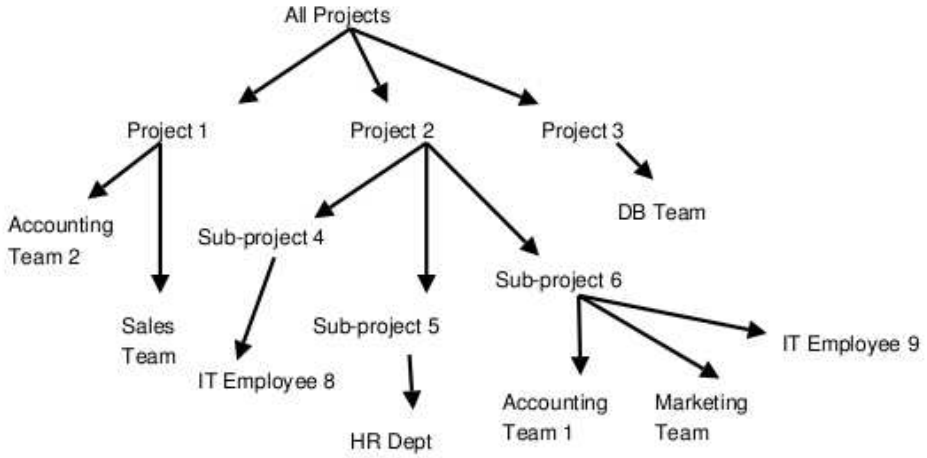
Introduction

Application-Specific Hierarchies

A large number of application-specific hierarchies exists.



(b) Application-specific geography hierarchy



(c) Application-specific project hierarchy

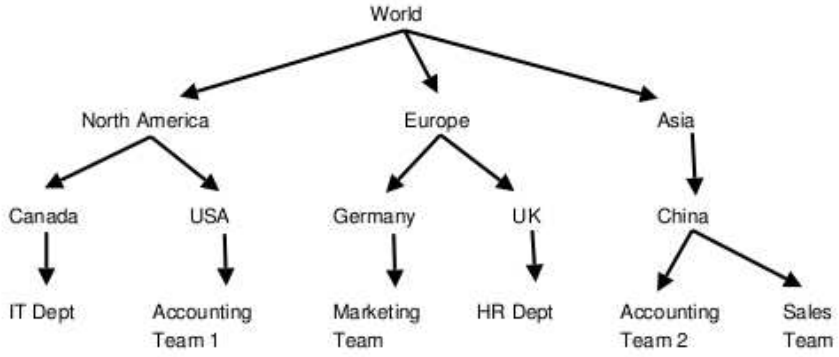
Introduction

Current Strategy

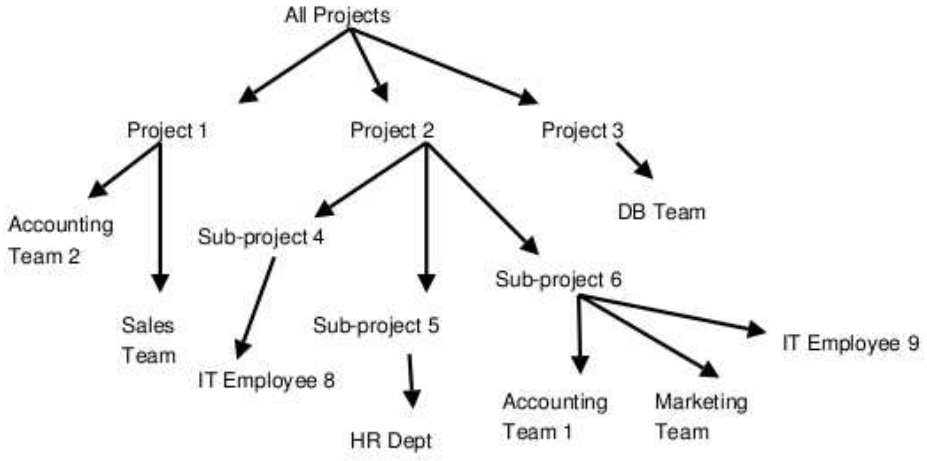
Pre-compute aggregates for some internal nodes of specific application hierarchies.

Problems with This Strategy

1. This pre-computation is unfeasible for all internal nodes of all application hierarchies.
2. OLAP query engines cannot exploit precomputed aggregates across hierarchies.



(b) Application-specific geography hierarchy



(c) Application-specific project hierarchy

Proposed Solution

Phase I – Overlap Discovery, Off-line

Discover & store overlapping relationships in the hierarchies of an OLAP environment.

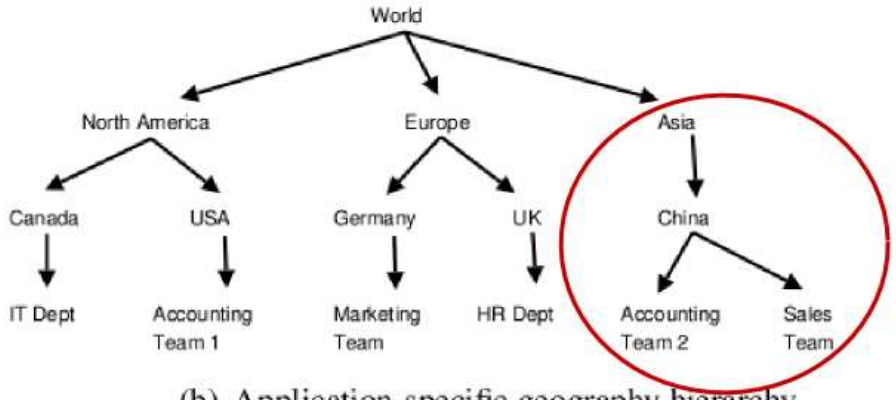
Input: A primary hierarchy. A set of application-specific hierarchies.

Output: A catalog table containing the overlapping information between hierarchies.

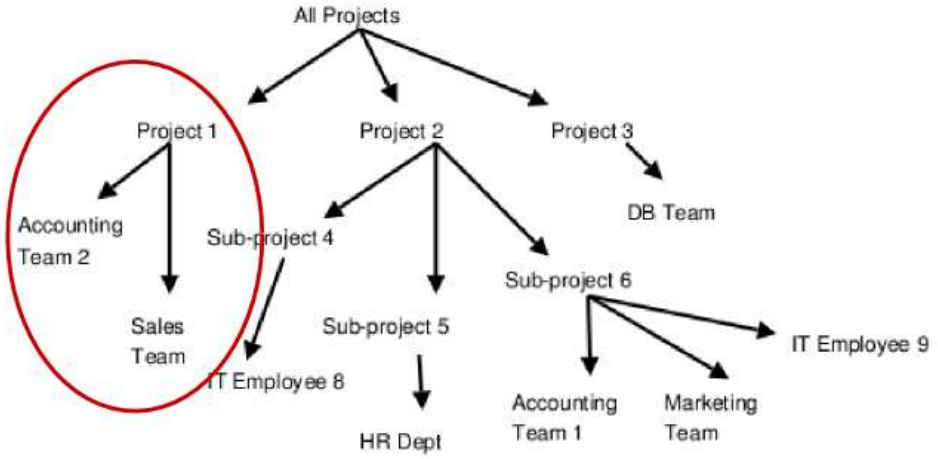
Example catalog table:

```
geography-hierarchy-Asia
:
```

```
project-hierarchy-Project1
:
```



(b) Application-specific geography hierarchy



(c) Application-specific project hierarchy

Proposed Solution

Phase II – Query Rewrite, On-line

Rewrite queries using the discovered relationships.

Input: An aggregation query. An application-specific hierarchy. The catalog table.

Output: An alternate query formulation.

Assumption: Catalog table stores the pre-computed overlapping sub-hierarchies.

Tasks: First, finds all overlapping sub-hierarchies shared with evaluated forest.

Second, alter the input query by replacing the cached sub-hierarchies.

Program Modules

Synthetic Forest Generating Module (from Dr. Lim)

Generates application-specific hierarchies as an XML file.

Overlapping Discovering Module (from Dr. Lim)

Returns a list of node pairs indicating the roots of maximal overlapping sub-hierarchies.

Query Rewriting Module

Checks for all possible rewrites and returns an altered aggregation query formulation.

Random Query Generating Module

Generates queries with specified size and overlapping against pre-computed aggregates.

Benchmark Module

Connects to a MySQL database, executes numerous queries, and collects analysis data.

Benchmarks Preparation

Create Table

```
CREATE TABLE db1.table1 (
  id INTEGER NOT NULL DEFAULT NULL AUTO_INCREMENT,
  name TEXT CHARACTER SET latin1 NOT NULL,
  value INTEGER NOT NULL,
  PRIMARY KEY (id))
ENGINE = MyISAM
ROW_FORMAT = DYNAMIC;
```

Populate Table

```
INSERT INTO table1 (name, value) VALUES ('L57', 36);
```

For a given catalog table, all leaves (e.g. `L57`) are inserted with dummy data (e.g. `36`).

For a given set of randomly generated queries, all necessary leaves are inserted.

More dummy data is inserted, and all data is shuffled to avoid sequential access.

Benchmarks Preparation

Populate Cache

```
SELECT value FROM table1 WHERE name = 'L57';
```

For a given catalog table

query all leaves for every shared sub-hierarchy,

aggregate `"value"`,

store the sum for each shared sub-hierarchy in memory.

Query Table and Record Timing Data

For a given aggregation query,

query all leaves,

aggregate `"value"`,

return the sum.

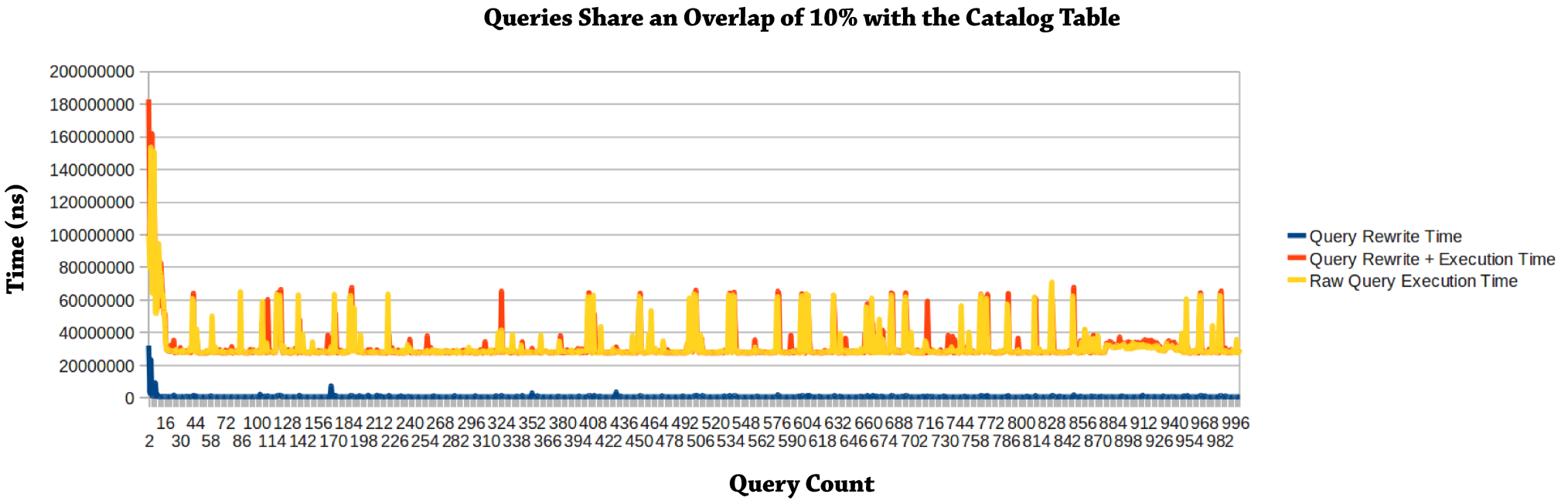
This is done on both the raw query and the re-written query.

Two results are verified to be the same.

Benchmark Results

Query Database

Queries are fixed in size and have ***small*** overlap with the catalog table.



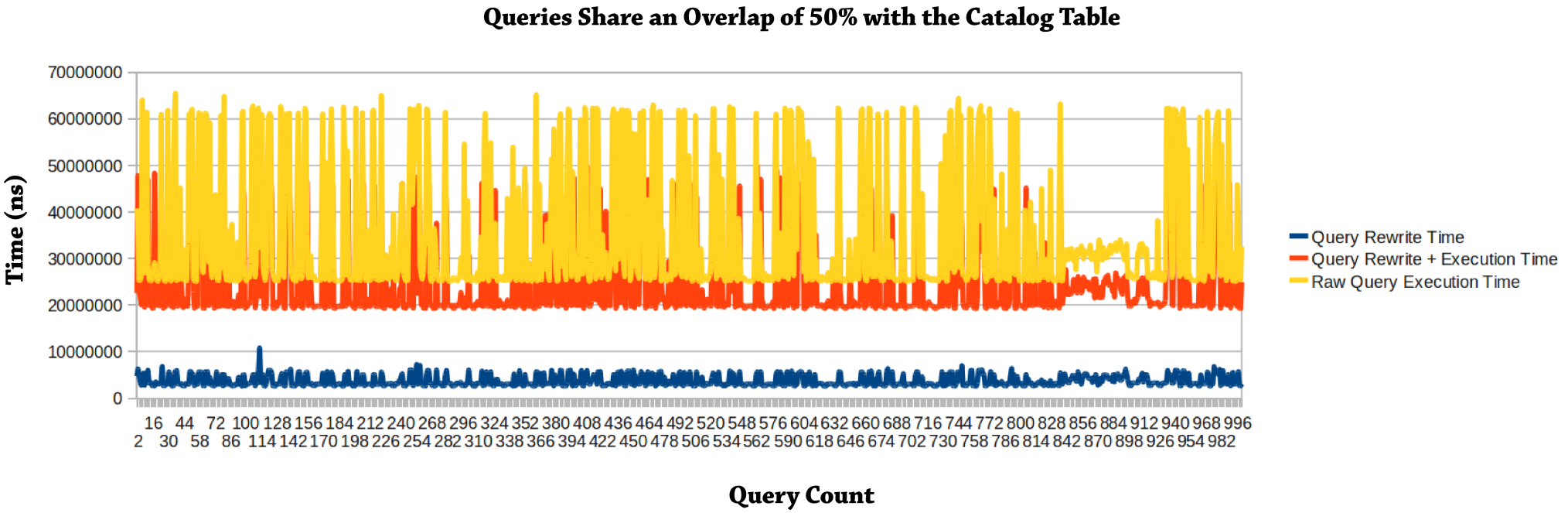
Re-writing queries does not dominate the query execution time.

Queries with a small overlap property do not take advantage of the catalog table.

Benchmark Results

Query Database

Queries are fixed in size and have ***medium*** overlap with the catalog table.



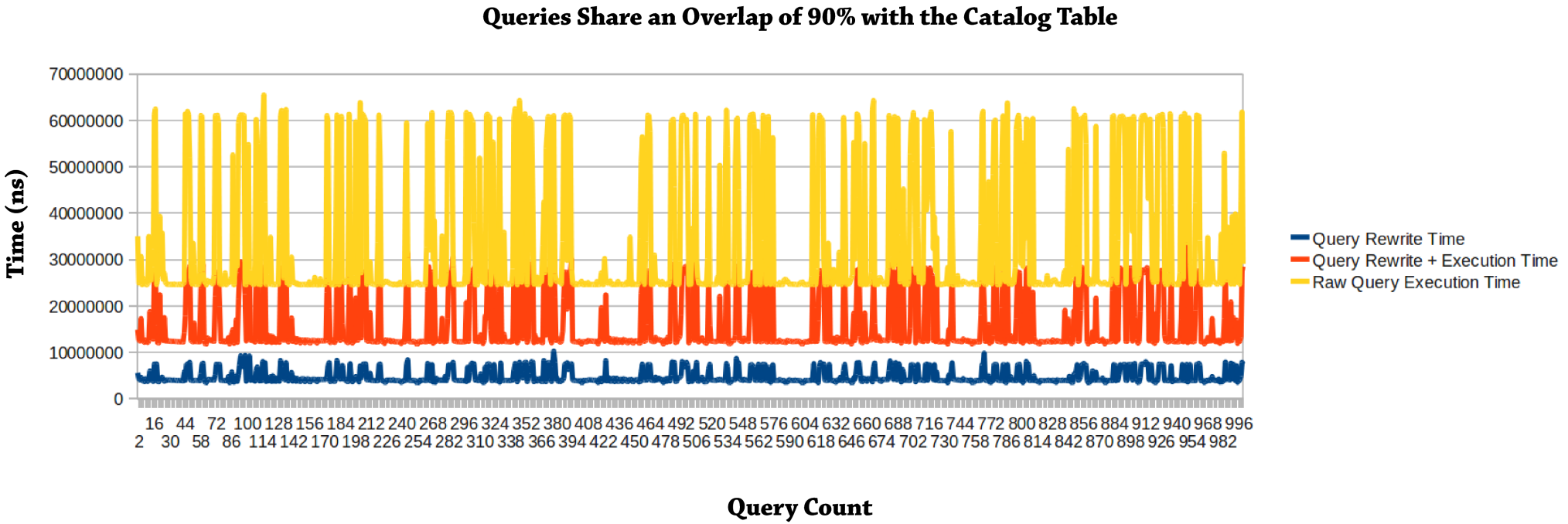
The benefit of re-writing queries becomes apparent.

Re-written queries executed noticeably faster.

Benchmark Results

Query Database

Queries are fixed in size and have ***large*** overlap with the catalog table.



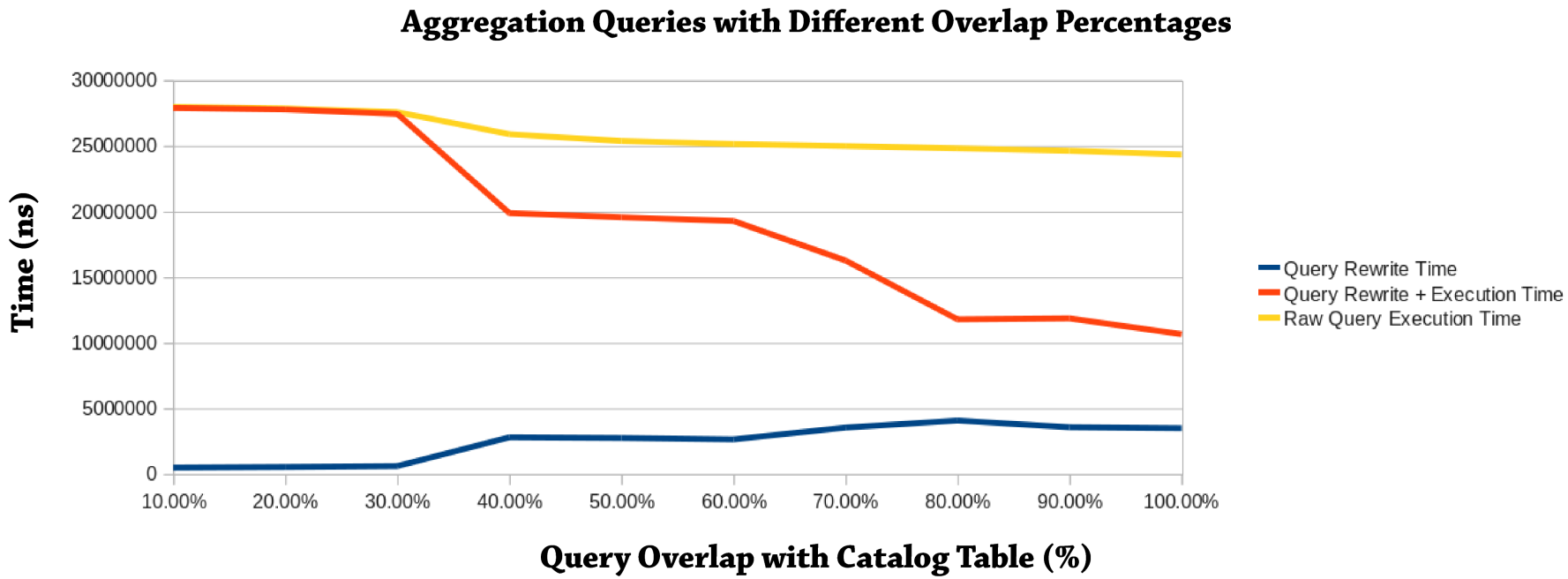
The re-written queries executed considerably faster than the raw queries.

The amount of time spent to re-write a query stays the same and does not dominate.

Benchmark Results

Query Database

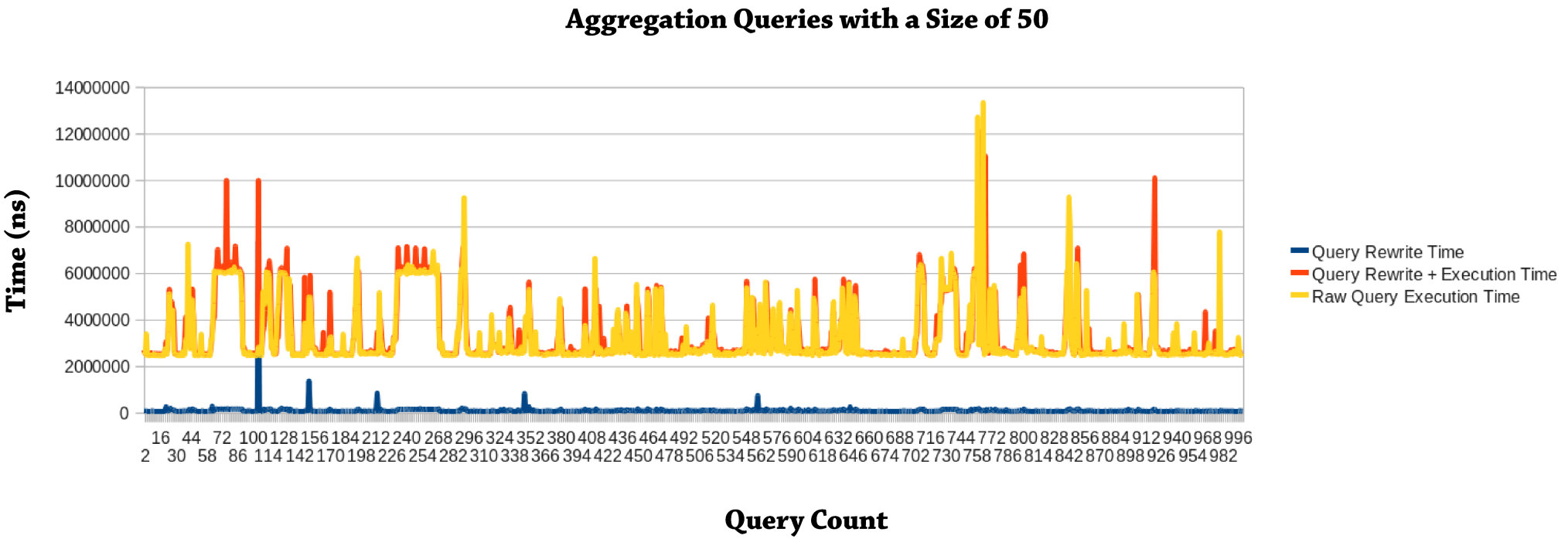
Queries are fixed in size and have ***varying*** overlap with the catalog table.



Benchmark Results

Query Database

Queries are fixed in overlapping percentage and have a ***small*** size.



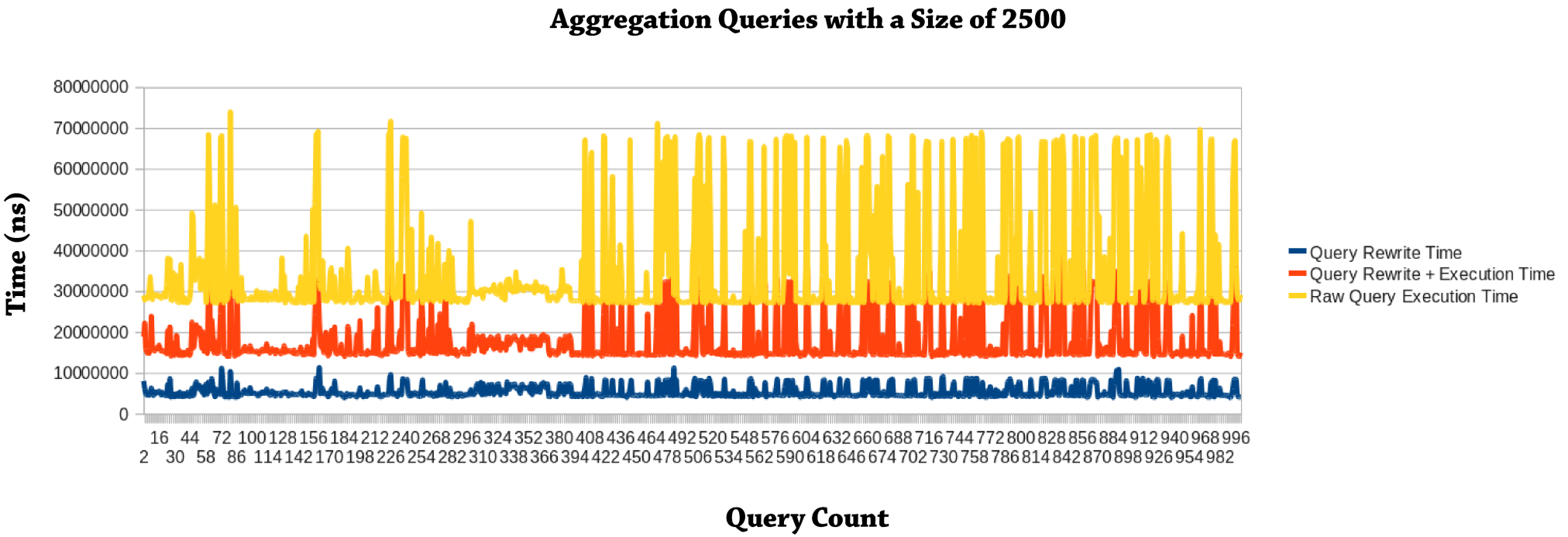
Re-writing queries does not dominate the query execution time.

Queries with small sizes do not take advantage of the catalog table.

Benchmark Results

Query Database

Queries are fixed in overlapping percentage and have a ***large*** size.

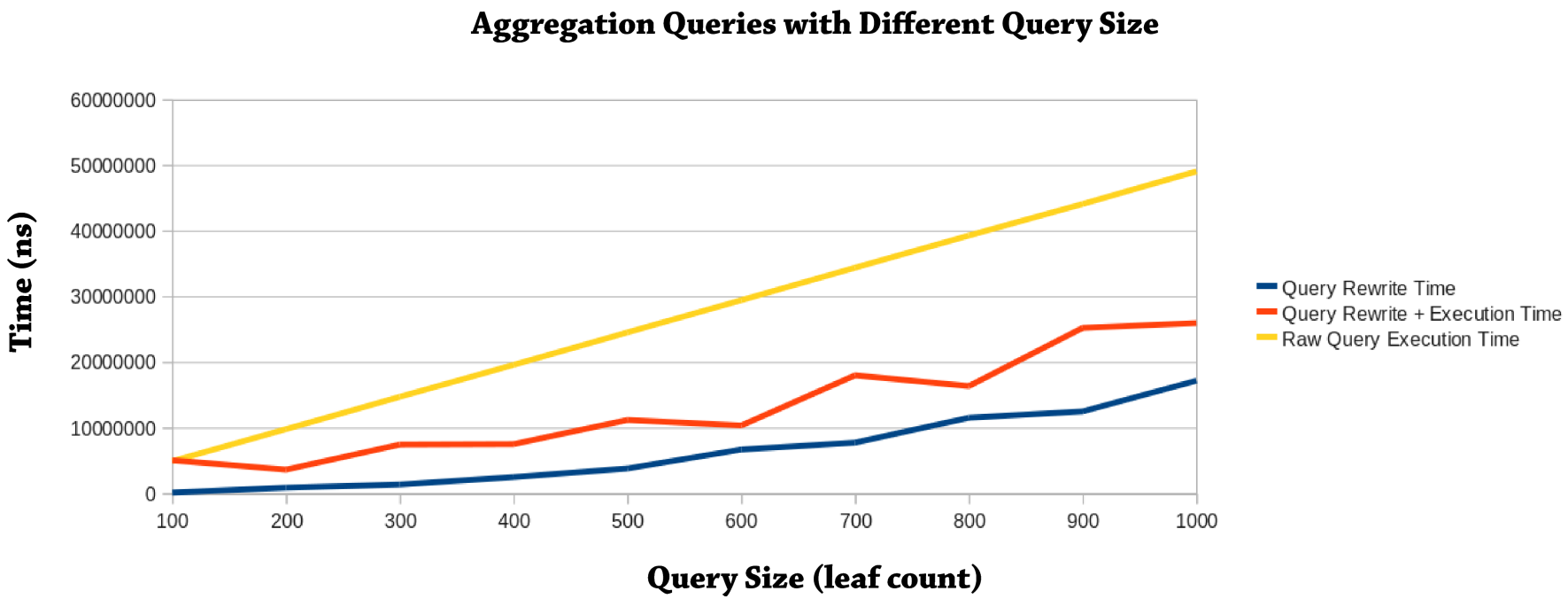


Query re-writing significantly improves the performance of large aggregation queries.

Benchmark Results

Query Database

Queries are fixed in overlapping percentage and have ***varying*** sizes



Conclusions

Problem Identified

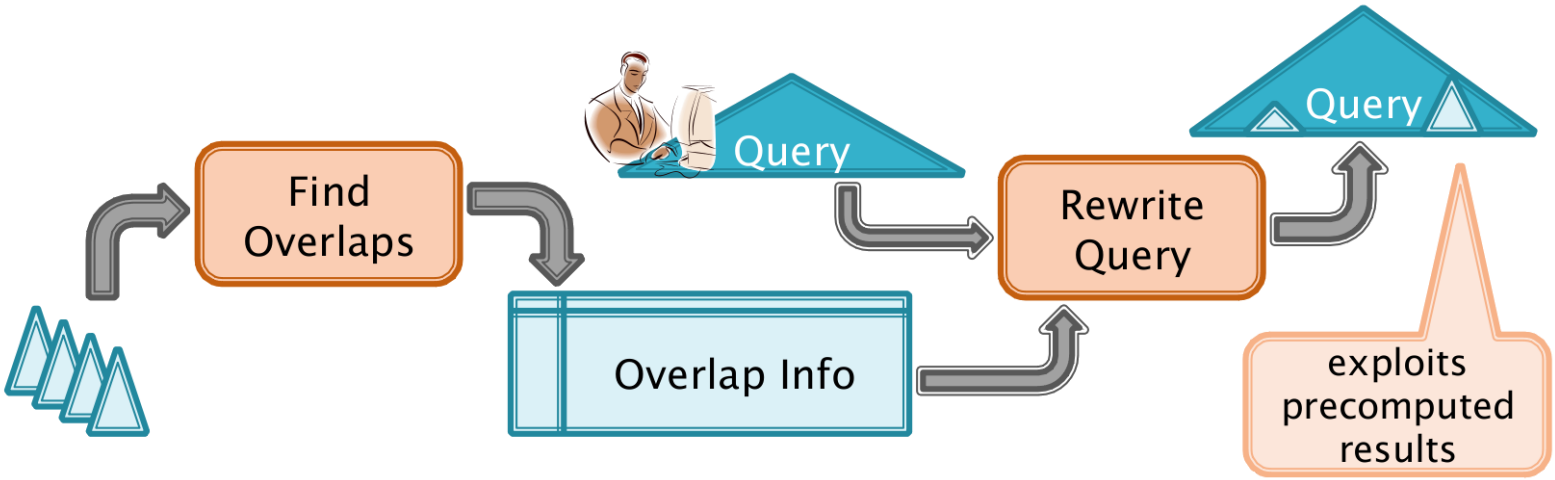
Uncontrolled proliferation of application hierarchies in enterprise data warehouses.

Inability of the OLAP query engines to exploit aggregates across different hierarchies.

Proposed Solution

Find hierarchy overlap information from an application-specific hierarchy forest.

Re-write aggregation queries to exploit overlaps and use pre-computed aggregates.



Conclusions

Previous Benchmark Work

Showed that finding overlap information is feasible.

Current Benchmark Work

Query re-writing does not dominate the execution time of aggregation queries.

Query re-writing significantly improves the performance of queries with large overlaps.

Query re-writing significantly improves the performance of large queries.

Possible Future Work

Perform end-to-end experimental evaluation with both phases.

Experiment on large forest with large catalog table that requires storage in database.

Improve the caching mechanism, e.g., the catalog caches intermediate sub-hierarchies.