Project Phase II

Bikash Roy
br8376@rit.edu
Rochester Institute of Technology
Rochester, NY

Ankit Jain
aj9761@g.rit.edu
Rochester Institute of Technology
Rochester, NY

Rishab Katta
rk4056@g.rit.edu
Rochester Institute of Technology
Rochester, NY

Milind Kamath
mk6715@rit.edu
Rochester Institute of Technology
Rochester, NY

ABSTRACT

A project write-up for Project Phase 2. In this Phase we're going to talk about a document-oriented model for our data-set and compare it with the relational model, Functional Dependencies and Normalizing of the initially loaded relations, the SQL queries providing detailed insights from our data-sets and indexing the columns in our relations to speed up the querying process.

KEYWORDS

Relational Model, non-relational model, Functional Dependencies, Normalization, Querying, Indexing

ACM Reference format:
Bikash Roy, Ankit Jain, Rishab Katta, and Milind Kamath. 2019. Project Phase II. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 3 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The data set of Daily Historical Stock Prices (1970 - 2018), we create a non-relational model, compare it with our relational model, Determine Functional Dependencies and Normalize the tables loaded in phase 1, Provide 5 SQL Queries that provide insights from our data-set and index selected columns to speed up the querying process. We also need to provide a python code to load the data set into the MongoDB database.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 LOADING DATA FROM RDBMS TO MONGODB

Code in python file.

3 NON-RELATIONAL MODEL AND COMPARISON WITH THE RELATIONAL MODEL

Non-Relational Model

company: {
_id : _, (auto generated by MongoDB)
ticker : _, (sort of like a "primary key")
exchange : _,
company_name : _,
sector : _, (id from sector collection. "foreign key")
industry : _ (id from industry collection. "foreign key")
}

historical_stock_price: {
_id : _, (serial integer. "primary key")
ticker : _, (like a "foreign key" to ticker of company)
open_price : _,
close_price : _,
adj_close_price : _,
low_price : _,
high_price : _,
volume : _,
stock_date : _
}

industry: {
_id : _, (bigserial. "primary key")
name : _,
}

sector: {
_id : _, (bigserial. "primary key")
name : _,
}

In the relational model, the values pertaining to each id are stored in a row(tuple). In the non-relational model, the same

values are stored as bson objects. In the relational model set of values for a particular id is called a row and set of rows is called a table. In non-relational model set of values for a particular id is called a document and set of documents is called a collection. Unlike relational model, We have a flexible schema in the Document-based model, meaning if we don't a value for a field, we can just not include it. So then different documents in the collection have different structure. NoSql databases offer BASE(Basically Available, Soft state and Eventual Consistency) compared to relational databases which offer ACID(Atomicity, Consistency, Isolation, Durability). We go for Non-relational databases when strong consistency is not required and we need to maintain data over a distributed network over many partitions and losing some data here and there won't matter.

4 QUERIES

Following are the 5 queries that derive information from the relational date base.

- Display company names and maximum percentage change that is greater than or equal to 15 and has least loss in the time period 2000-2018.

  SELECT DISTINCT company-name, MAX((((closeprice - adjcloseprice) / closeprice) * 100)) AS MaxPercentageChange FROM company JOIN historicalstockprice ON historicalstockprice.ticker = company.ticker WHERE CAST(stockdate AS varchar) > '2000-01-01' AND CAST(stockdate AS varchar) < '2018-12-31' AND (((closeprice - adjcloseprice) / close-price) * 100) >= 15 GROUP BY companyname ORDER BY MaxPercentageChange;

  Execution Time Without Index: 8.192
  Execution Time With Index: 7.257

- Display company name, average of opening prices until 2018 which are greater than 30 dollars and where company name includes "Limited" or "Inc" in their names.

  SELECT DISTINCT companyname, avg(openprice) AS AvgOpenPrice FROM company JOIN historicalstockprice ON historicalstockprice.ticker = company.ticker WHERE CAST(stockdate AS varchar) > '1980-01-01' AND CAST(stockdate AS varchar) < '2018-12-31' AND companyname ilike '%Limited' or companyname ilike '%inc' GROUP BY companyname
  HAVING AVG(openprice) > 30;

  Execution Time Without Index: 1.840
  Execution Time With Index: 1.508

- Display company name, sector name, difference in opening and closing price of all technological companies where difference is less than 0.02.

  SELECT companyname, sec.name, hist.highprice - hist.lowprice AS diffInPrediction FROM company AS comp JOIN sector AS sec ON comp.sector = sec.id JOIN historicalstockprice AS hist ON comp.ticker = hist.ticker WHERE sec.name ilike 'Technology' AND hist.highprice - hist.lowprice < 0.02 ORDER BY companyname;

  Execution Time Without Index: 1.281
  Execution Time With Index: 2.17

- Display the names of companies and volumes of shares dumped under health-care sector, sold by NASDAQ exchange where loss is greater than 50 percent.

  SELECT companyname, volume FROM company AS comp join sector AS sec ON comp.sector = sec.id JOIN historicalstockprice AS hist on comp.ticker = hist.ticker WHERE comp.exchange ilike 'NASDAQ' AND sec.name ilike 'Health Care' AND (((closeprice - adjcloseprice) / closeprice) * 100) >= 50 ORDER BY volume desc;

  Execution Time Without Index: 1.272
  Execution Time With Index: 0.327

- Display companies with its maximum loss along with its maximum loss of amount in descending order for Integrated Oil Companies.

  SELECT DISTINCT comp.companyname, MAX(hist.closeprice - hist.openprice) AS MaxLoss , ((MAX(hist.closeprice - hist.openprice))*hist.volume) AS MaxAmountLoss FROM industry AS ind JOIN company AS comp ON ind.id = comp.industry JOIN historicalstockprice AS hist ON hist.ticker = comp.ticker WHERE CAST(hist.stockdate AS varchar) > '2015-01-01' AND CAST(hist.stockdate AS varchar) < '2018-01-01' AND ind.name ilike 'Integrated Oil Companies' GROUP BY comp.companyname, hist.volume ORDER BY MaxLoss desc;

  Execution Time Without Index: 3.324
  Execution Time With Index: 0.214

5 INDEXING CODE

1) CREATE INDEX tickeridx ON company (ticker);
2) CREATE INDEX tickerhistidx ON historicalstockprice (ticker);
3) CREATE INDEX companynameidx ON company USING GIN (companyname gintrgmops)
4) CREATE INDEX sectornameidx ON sector USING GIN (name gintrgmops)
5) CREATE INDEX companyexchangeidx ON company USING GIN (exchange gintrgmops)
6) CREATE INDEX industrynameidx ON industry USING GIN (name gintrgmops)

The attributes have been selected where it has been queried the most, searched upon for a company name, sector name or a particular industry etc.

6 FUNCTIONAL DEPENDENCIES AND NORMALIZATION

Code in python file.
We need Normalization in our relational database because we need to avoid any insertion, deletion and updation anomalies in our database. We followed the pruning approach to determine the functional dependencies, wherein we partition the rows in our table based on similar values and check for refinement on each column value. python code for the pruning approach is included in the python file stocksphase2.py.

After running the pruning approach for the table company, the program returned the following functional dependencies:

$ticker \rightarrow \{exchange, companyname, sector, industry\}$

and on running the same code on historical_stock_price,sector and industry the program returned the following functional dependencies respectively:

$id \rightarrow \{ticker, openprice, closeprice, adjcloseprice, lowprice, highprice, volume, stockdate\}$
$id \rightarrow \{name\}$
$id \rightarrow \{name\}$

This means that all the tables in our database are already Normalized. However based on the instructions provided in Phase 1 of the project, we modified the initial structure of our relational model. We now moved the sector names to a separate table and we created serial ids for those name. Same for the industry names as well. The original company table references the sector ids and industry ids from sector and industry column respectively.