**Merchandise Recommendation System with Relational Database and Graph Databases**

**Group 6**

Zuge Jin

Xiaozheng He

Qian Mai

Kaicheng Yang

**CSCI 5559 Database Systems**

***Computer Science and Engineering***

***University of Colorado Denver***

[***1.***](#_gjdgxs) ***Problem Statement and Background***

Merchandise Recommendation System is a kind of information filtering system that can predict which merchandises users may need or purchase in future and recommend those merchandises to users. Merchandise Recommendation System is more and more popular because it can bring benefits to both retailers and customers. Retailers can have a good acknowledge of the consumption habit of customers and improve their sales performance; customers can have an acknowledge about other merchandises they may need after they purchase something and make their consumption easier. Today, Merchandise Recommendation System is used in a variety of areas and get success.

In this project, we implement a Merchandise Recommendation System by using three distinct databases: relational database(clustered and unclustered), Neo4j and OrientDB. The data we use is found online which is a sales records of an online retailer during the whole 2011. After the implementation, we compare the performance of those three databases. The performance measures data storage, data import time, execution time, records of recommendation returned and recommendation depth. Data storage is the usage on disk to store the data; data import time is the execution time of import all data into database; the execution time is how long the system perform the execution; the records of recommendation are the merchandises system want to recommend to users; the recommendation depth indicates what level is the recommendation at, for example, there is an association like A--B--C, then product C is the second depth recommendation of product A. After the comparison, we show the result and make an analysis why we get that result.

[***2.***](#_30j0zll) ***Related Work***

According to the resources on line, the performance of Graph DBMS is higher than Relational DBMS on workload that return the neighbor vertices of a root vertex. And according to the paper “XGDBench: A benchmarking platform for graph stores in exascale clouds” published in Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference, the performance of OrientDB is better than Neo4j on workload that return all neighbor vertices and their attributes of a vertex V. The figure 2.1 shows the performance. However, they only used a large amount of data to test the performance and they only test one depth of neighbor vertices of the root vertex. So we don’t know whether the performance changed for small data or deeper depth of neighbor vertices. In our project, we test those conditions for both Neo4j and OrientDB and analyse the result.

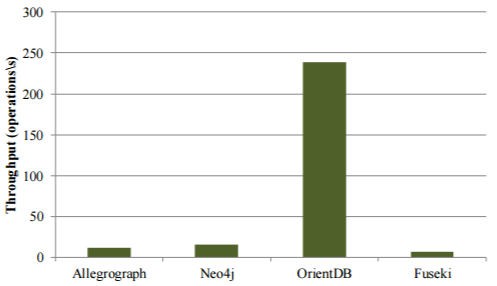


Figure 2.1 The performance of return neighbor vertices

[***3.***](#_1fob9te) ***Solution***

[**3.1**](#_tyjcwt) **Raw Data Preprocessing**

[**3.1.1**](#_tyjcwt) **Data Importation**

Because Microsoft SQL Server 2012 have a friendly user interface, we decide to import data into it to improve the accessibility of the row data. We create a new table with similar column format as the raw resource.

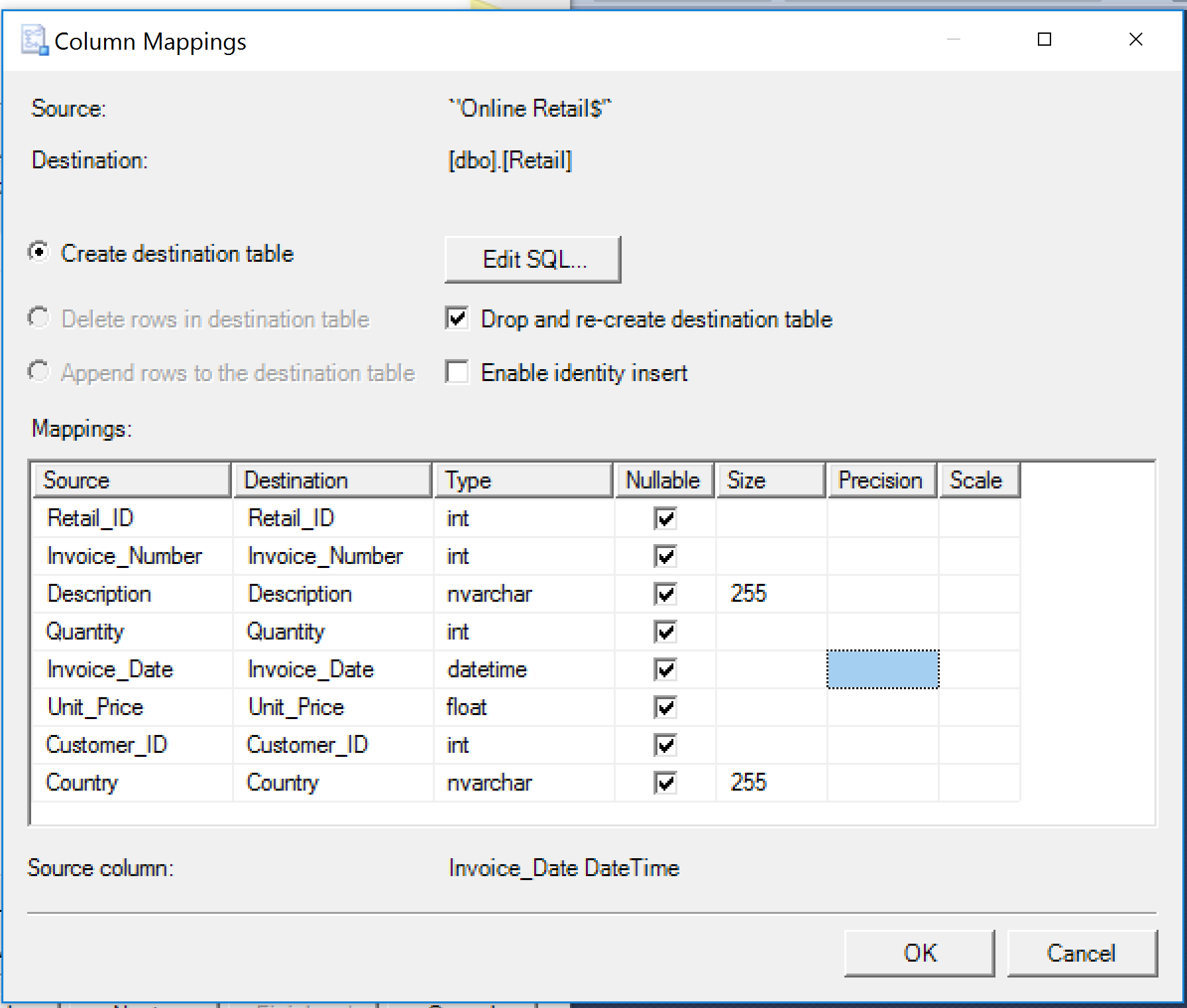


Figure 3.1 Retail Table Importation

The original data are stored as xlsx files. SQL Server 2012 Import Wizard is capable to import xls file. However, the excel(xls) file has a row limitation so we have to separate the data into multiple files then import it into SQL DB respectively.

[**3.1.2**](#_tyjcwt) **Data Screening**

During the association procedure, we find that the some record of the description contains invalid data such as return items. We remove those invalid record by filtering the purchase transaction quantity.

Some of the description name contains special character that may influent creating associations with other products. We use sql replace function to remove special character from the column. (replace([column], 'character', '') ).

Eventually, we still have more than 500k valid records.

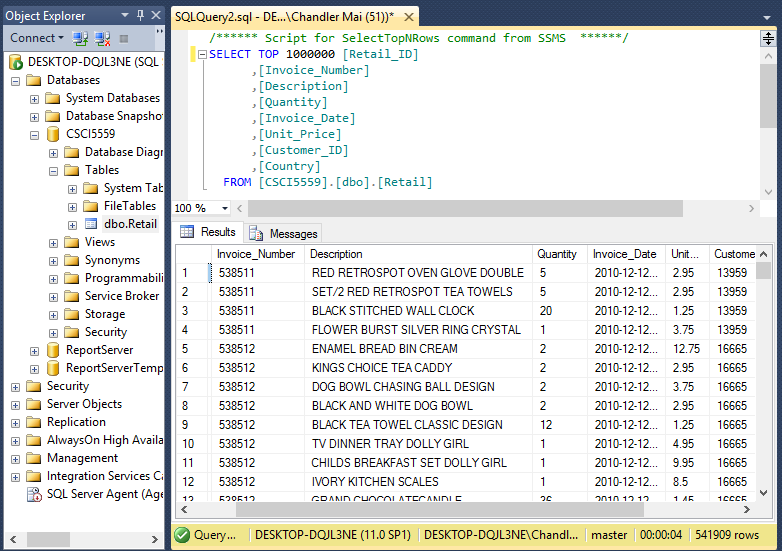
****

Figure 3.2 Retail Table Screening

**3.1.3 Association Establishment**

The recommendation bases on the association between items. According to the previous recommendation research, without the item review scores, we create the associations between every two items of all items per customer purchase per day (using the same invoice number). Hence, the associations are the combination two selection of those purchase items: .

Since there are huge item records, we have to create a stored procedure to generate their association. We gather the invoice number list. Then use nested loop to create association for each two item with the same invoice number. The outcome data has been avoided permutation redundancy (A|B and B|A).

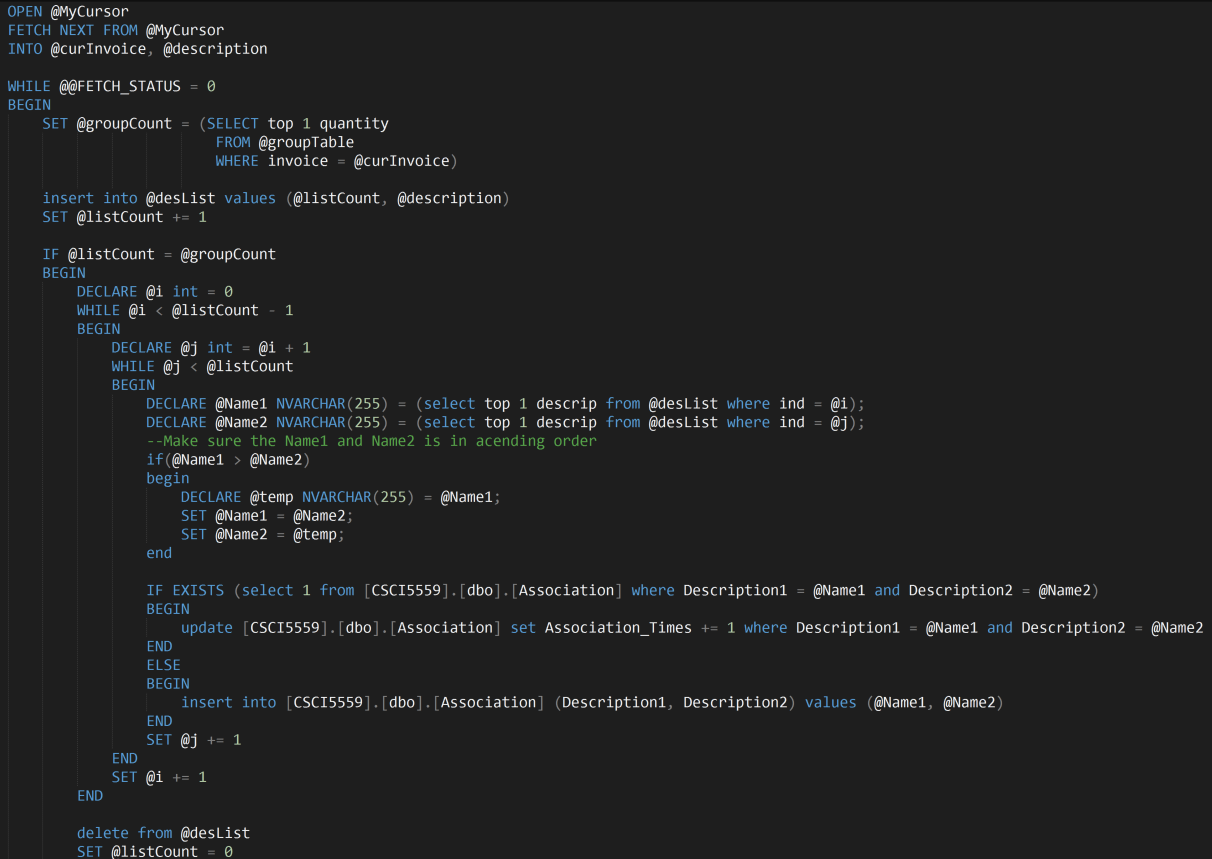


Figure 3.3 Get Association Stored Procedure

After three day’s procedure running, we have successfully generated almost one million partial association records.

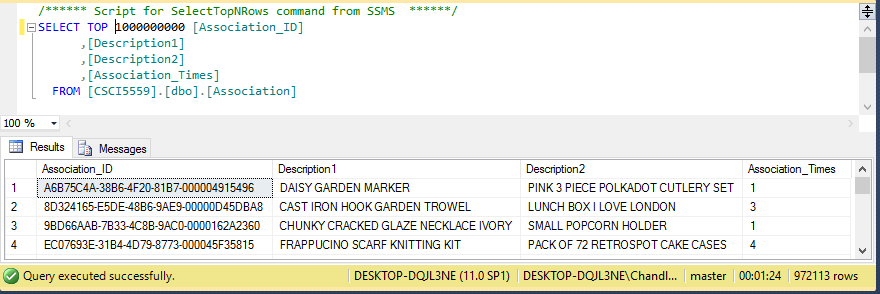


Figure 3.4 Association Record Count

**3.1.4 Test Object Selection**

For getting the best demonstration, we create a table for counting the recommendation in each depth for each items, and iterate another stored procedure in the following SQL server section. Then select a best test item object that contains three level recommended items. Depth 1: 6 records; depth 2: 1958 records; depth 3: 2375 records. This record will be used for accuracy test in the following three databases recommendation results.

[**3.2**](#_tyjcwt) **SQL Server Implementation**

**3.2.1 Clustered Index Structure Recommendation**

Since the association has set Association\_ID as primary key. So it is a clustered index structure.

The recommendation procedure iterates all first relevant items and find the their relevant items… and so on. The search strategy is Breadth First Search (BFS).

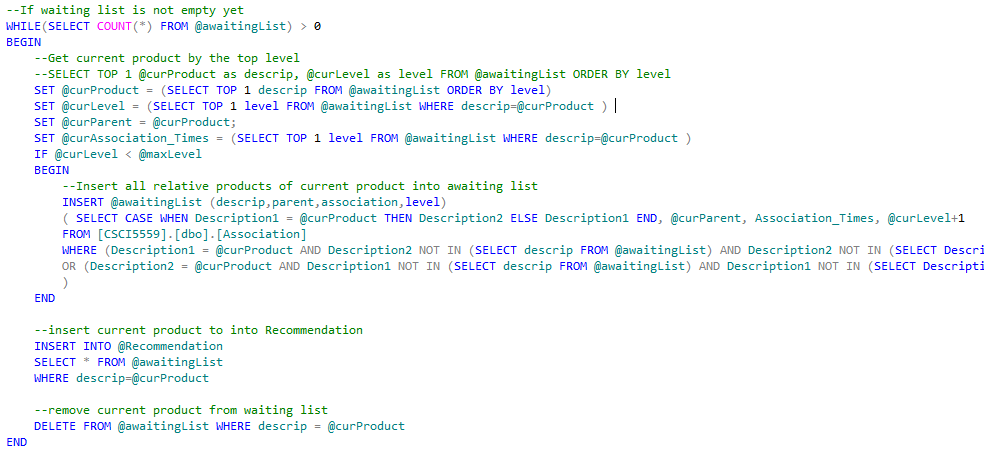


Figure 3.5 Get Recommendation Stored Procedure

The output format contains parent, association times and depth.

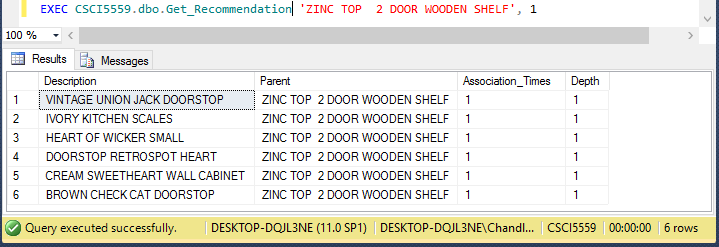


Figure 3.6 Recommendation Properties

**3.2.1 Unclustered Index Structure Recommendation**

To create an unclustered table, we just need to make a copy of the empty clustered association table. Then drop the primary key index and create a nonclustered index.

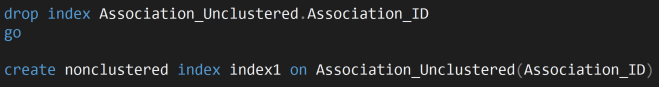
****

Figure 3.7 Clustered to Unclustered Table

After the structure has been defined, we can re-import the records from the clustered table directly.



Figure 3.8 Unclustered Table Data Importation

Run the same get\_recommendation stored procedure to compare the runtime with the clustered results.

[**3.3**](#_tyjcwt) **Neo4j Implementation**

Neo4j is the most popular Graph DBMS developed by Neo Technology. So we choose it as our first

Graph DBMS to apply the Merchandise Recommendation System since it is a widely used graph database.

Firstly, we need to import data into Neo4j. The data we used is the Associations.csv file with 972K pairs of associations with association times. In order to process importing and querying data on Neo4j database, the syntax and usage of Cypher language which is the declarative graph language of Neo4j need to be grasped.

After launching Neo4j, a web page need to be loaded by browser to manipulate the database. Neo4j can import data simply through a bunch of Cypher statements which is very intuitive. The data in Neo4j is organized in an atomic way which exist as many entities with properties related to them. There are basically two kind of objects, node and relationship. Nodes have label as well as relationships has type to classify them, thus, nodes with same label and relationships with same type have identical properties names. We loaded the items as nodes with label “Items” and property “Description” , the associations are loaded as relationships with type “Association” and property “weight” connecting these nodes. The nodes and relationships are created at the same time loading Associations.csv by “merge” statement which will create new nodes when not exists and skip to directly create relationships when the nodes already exist. Part of the nodes and relationships are shown below in Figure. We can find that all the 972113 relationships are created by looking at the result of query in Figure.

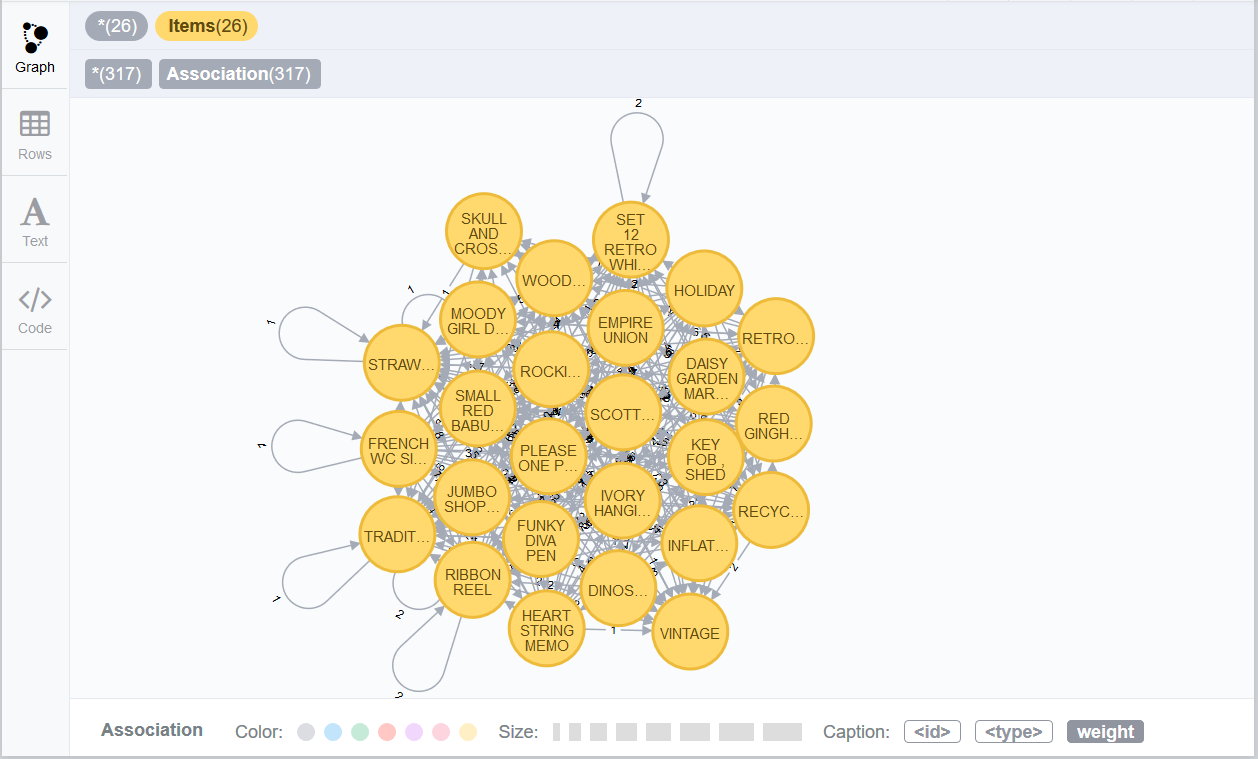
 

Figure 3.9 Thumbnail of nodes and relationships in Neo4j Figure 3.10 Number of Association in Neo4j

The query of traverse the nodes connect to a certain node with assigned depth in Neo4j is intuitive as tracking in the graph, it will automatically avoid “going back” along the same path. In that way the result returned is the same as the result of relational database. The nodes and relationships from query result of item *'ZINC TOP 2 DOOR WOODEN SHELF'* with depth=1 are shown below.

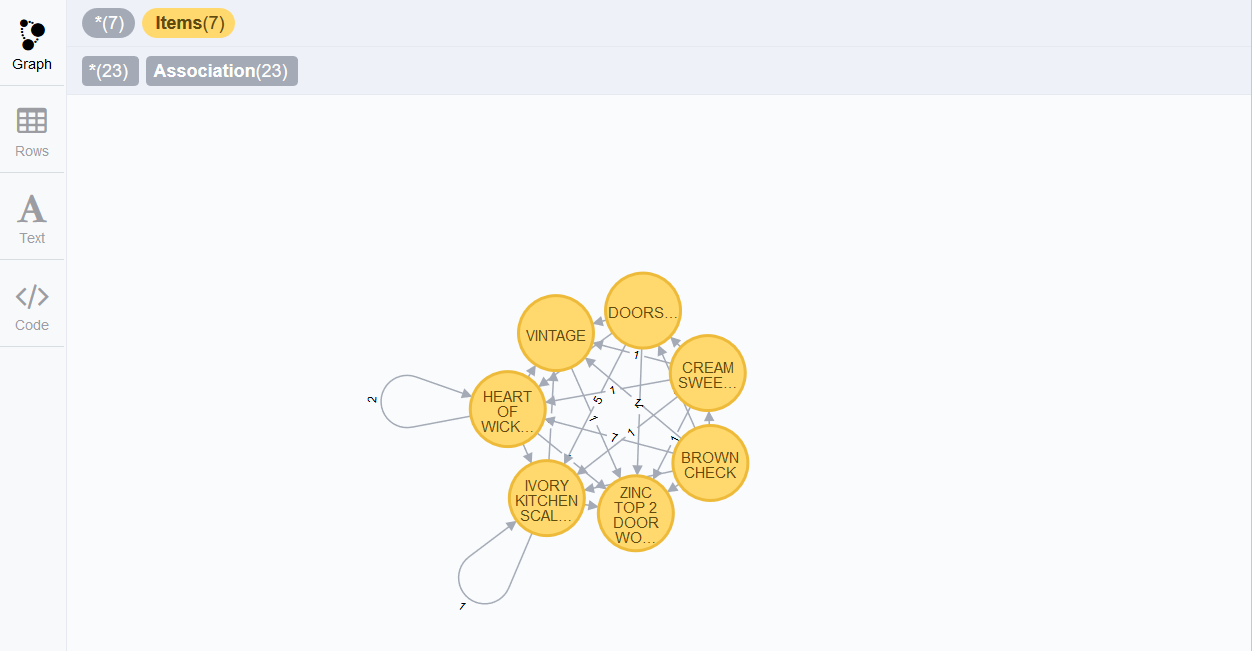


Figure 3.11 *'ZINC TOP 2 DOOR WOODEN SHELF'* with depth=1 in Neo4j

[**3.4**](#_tyjcwt) **OrientDB Implementation**

OrientDB is Multi-model DBMS (Document, Graph, Key/Value) developed by OrientDB LTD, we choose it to apply our Merchandise Recommendation System since it has the top second rating among other graph database and it has a different mechanism of processing graph data.

First, download and extract OrientDB by selecting the appropriate package provided on OrientDB download website. Start the server by running the server.bat (Windows System) scripts located in the bin folder. Once OrientDB is running, enter the following URL in a browser window: http://localhost:2480. This is the Studio which is a web tool for Databases.

In order to import our Associations data into the OrientDB to test the query performance, ETL (Extractor Transformer and Loader) which is a module for OrientDB provides support for moving data to and from OrientDB databases using ETL processes is needed. Basically, the process of importing using ETL is to call the oetl.bat located in bin directory by command line to run the specific JSON file. In this JSON file, we should state the directory the CSV file belongs to, which classes the vertices and edges belong to, which folder we need the database files to be generated to. Because the CSV file has three column, two of them are the descriptions of the associated items, one of them is the association times which is expected to be loaded as “weight” of associations, therefore, how the two columns are read by ETL and how two vertices are connected need to be fully considered and included in that JSON file as well. Figure below is the Associations.json file we used.

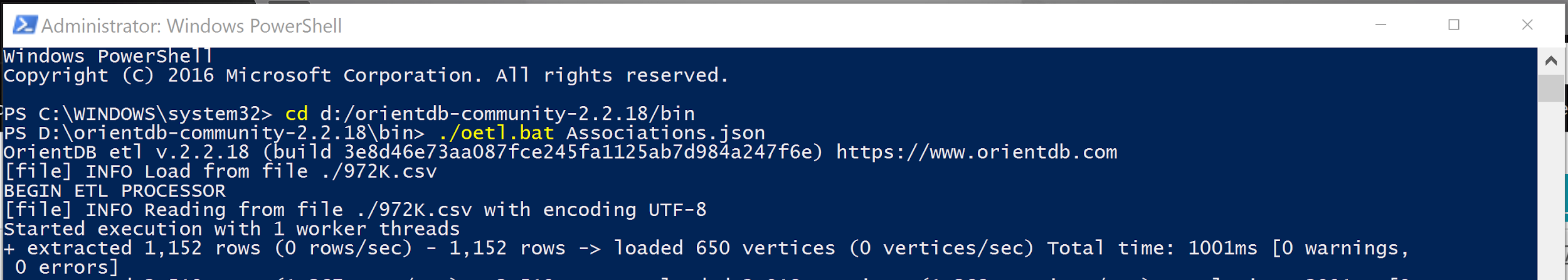


Figure 3.12 Associations Import Commands



Figure 3.13 Associations.json

After importing the data as vertices and edges successfully, there are 2357 vertices with property “Description” and 972113 edges with property “weight” connecting these vertices. Figure shows the schema of the database.

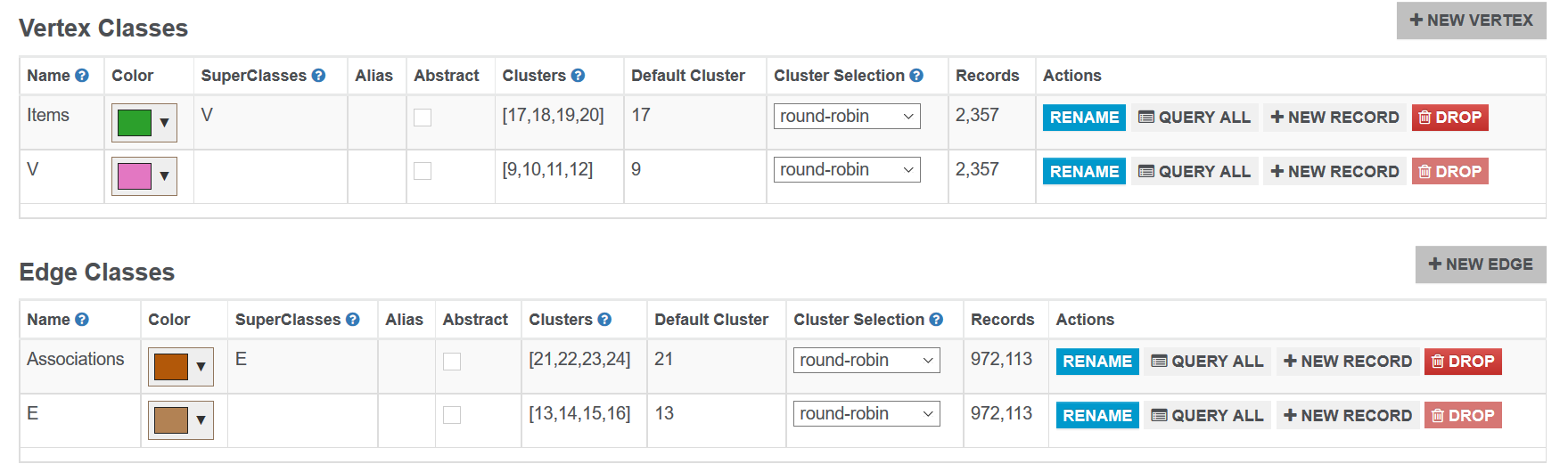


Figure 3.14 OrientDB Schema

Then we could run test cases on this database. The vertices and edges from query result of item *'ZINC TOP 2 DOOR WOODEN SHELF'* with depth=1 are shown below.

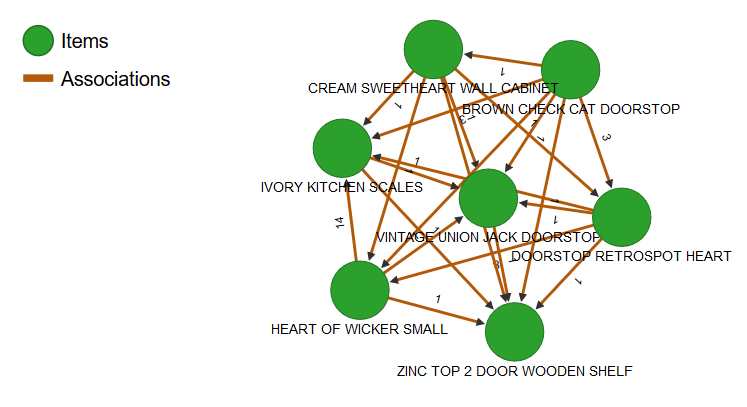


Figure 3.15 *'ZINC TOP 2 DOOR WOODEN SHELF'* with depth=1 in OrientDB

***4. Results***

**4**[**.1**](#_tyjcwt) **Data Storage Comparison**

**4.1.1 RDBMS Clustered vs Unclustered**

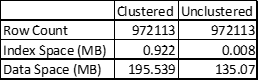
****

Figure 4.1.1 RDBMS Clustered vs Unclustered Storage

As Figure 4.1.1 shows above, the usage on disk of Unclustered RDBMS is 2/3 of the usage on disk of Clustered RDBMS. When we cluster the data, there are at most 2/3 of space can be used to store the data in case to insert new data, however, 100% space can be used for unclustered case. So the usage of clustered RDBMS should be 1.5 times of unclustered RDBMS.

**4.1.2 Neo4j vs OrientDB**

****

Figure 4.1.2 Neo4j vs OrientDB on Storage

The usage of OrientDB is more than Neo4j on our data scale. It may changed if the scale of data is huge.

**4**[**.2**](#_tyjcwt) **Data Import Time Comparison**

****

Figure 4.2 Neo4j vs OrientDB on Import Time

OrientDB need more time to import this scale data into database. It may changed if the scale of data is huge.

**4.3 Recommendation Search Time Comparison**

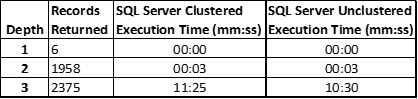
****

Figure 4.3.1 Execution Comparison of Clustered and Unclustered of RDBMS

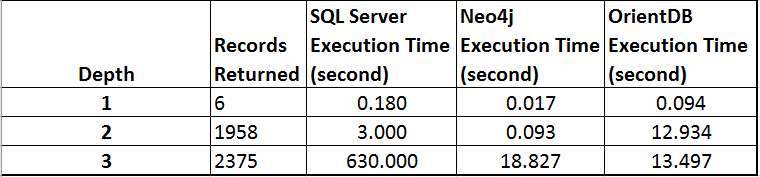
****

Figure 4.3.2 Execution Comparison of RDBMS and Graph DBMS

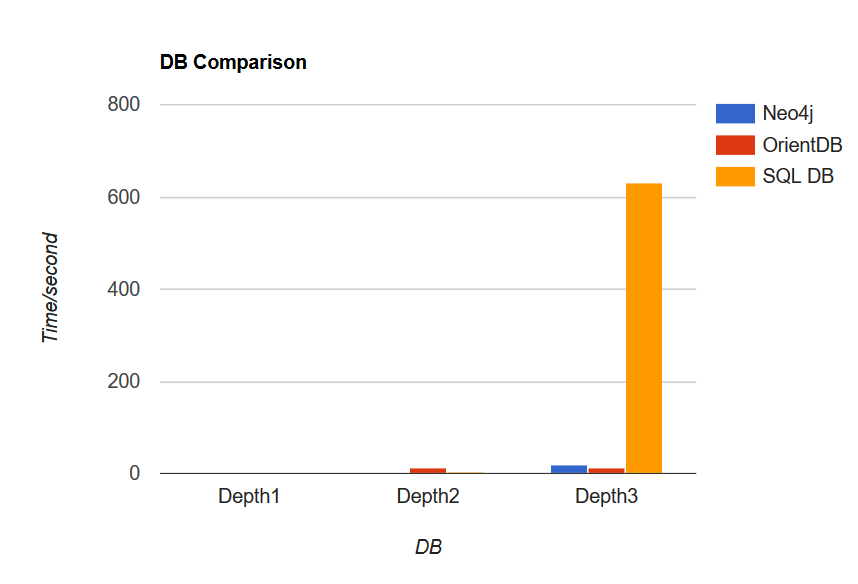
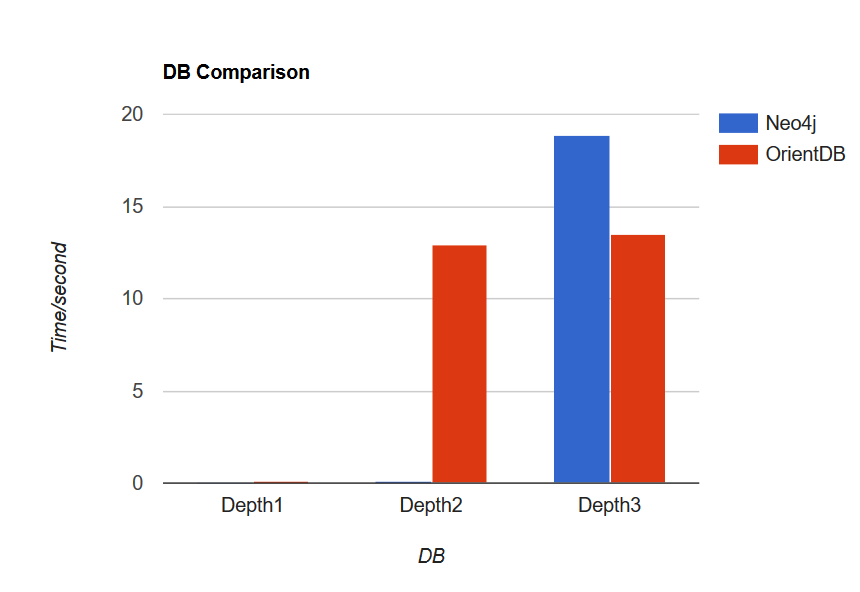
 

Figure 4.3.3 Comparison among Neo4j, OrientDB, SQL Server Figure 4.3.4 Comparison between Neo4j and OrientDB

From the results shown above, we found that clustered data consumes more data pages than unclustered data but consumes less time than unclustered data. According to the comparison among different databases on recommendation search, we found that the processing time of SQL Server is the longest among different depths of test cases. As for graph databases, OrientDB performs better than Neo4j on depth=3 while Neo4j performs better on depth=1 and depth=2 which involved less data than depth=3 test case. Besides, Neo4j takes less time to import data.

According to the analysis of our results, OrientDB is a distributed database and it uses physical pointer instead of B+tree indexing, that explains why it can search faster than Neo4j on large datasets. At the same time, OrientDB is also a multi-model database unlike Neo4j which is a graph database only, it may cost more to construct data in OrientDB.

***5***[***.***](#_4i7ojhp) ***Conclusions and Future Work***

According to results, the import time of OrientDB is faster than Neo4j. Graph DBMSs(Neo4j and OrientDB) have the higher performance than RDBMS for all depth. Uncluster RDBMS has higher performance than Cluster RDBMS and the execution time increases rapidly when the depth is increasing. For the comparison between Neo4j and OrientDB, the result shows Neo4j performs better than OrientDB if the depth is low(depth <= 2) or the data scale is small; but for large scale data(more vertices and relationships) or deeper depth(depth >= 3), OrientDB has better performance than Neo4j.

The future work is focus on the Graph DBMS recommendation optimization. We will filter the association by the weight value to improve the recommendation accuracy. The weight is the frequence of two products are on the same invoice.

***6. Team Contribution***

1. **ZUGE JIN**

Project OrientDB Implementation/Evaluation Charger

Verify raw data; Establish database environment; Import data objects and their relationships; Design and optimize queries; Collect and analyze output data results; Engage final report documentation;

1. **XIAOZHENG HE**

Project Neo4j Implementation/Evaluation Charger

Collect raw data; Establish database environment; Import data objects and their relationships; Design and optimize queries; Collect and analyze output data results; Engage final report documentation;

1. **QIAN MAI**

Project SQL Server Implementation/Evaluation Charger

Process raw data; Establish database environment; Import data objects and their relationships; Design and optimize queries; Collect and analyze output data results; Engage final report documentation; Organize the group meeting.

1. **KAICHENG YANG**

Project Theory/Report Charger

Project background knowledge study; Process raw data; Output final data diagrams and charts; Compare performances of databases; Engage final report documentation and presentation; Organize overall documents architecture.

**Evaluation of how well these responsibilities were carried out (1--10, 10 is highest):**

1. \_\_\_\_10\_\_\_\_ 2. \_\_\_\_10\_\_\_\_ 3. \_\_\_\_10\_\_\_\_ 4. \_\_\_\_10\_\_\_\_

**Evaluation of each group member's overall performance (1--10):**

**Effort:** 1. \_\_\_10\_\_\_\_\_ 2. \_\_\_\_10\_\_\_\_ 3. \_\_\_\_10\_\_\_\_ 4. \_\_\_\_10\_\_\_\_

**Results:** 1. \_\_\_10\_\_\_\_\_ 2. \_\_\_\_10\_\_\_\_ 3. \_\_\_\_10\_\_\_\_ 4. \_\_\_\_10\_\_\_\_

**Evaluation of the contribution of each group member to the total project (percentage, total should = 100):**

1. \_\_\_\_25\_\_\_\_ 2. \_\_\_\_25\_\_\_\_ 3. \_\_\_\_25\_\_\_\_ 4. \_\_\_\_25\_\_\_\_

**We bond as a powerful and innovative team with everybody bring the best endeavor on this project.**