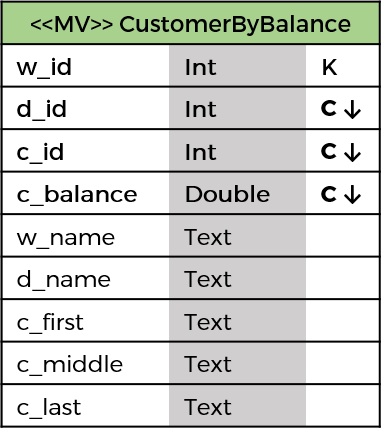
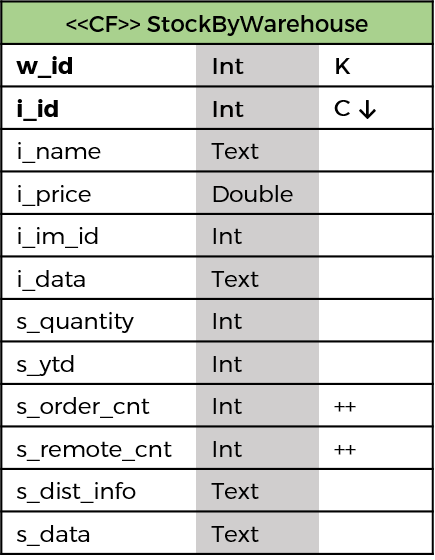
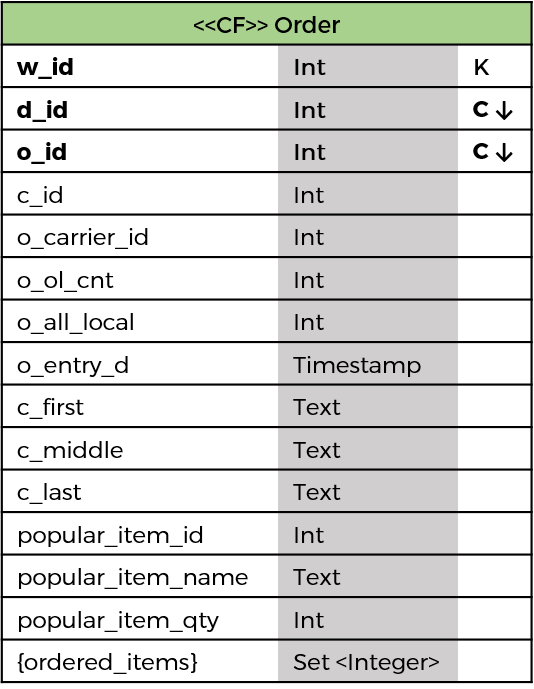
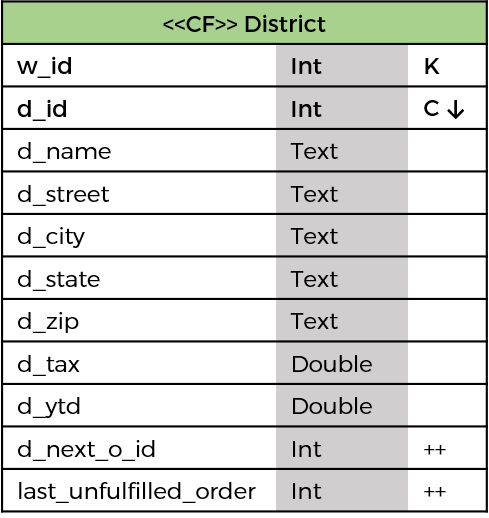
CS4224: Distributed Database

Team 6

|  |  |
| --- | --- |
| Lai Mun Keat | A0129561A |
| Zhang Chuanqi | A0175398L |
| Xiao Yuxin | A0131334W |
| Zhou Fanyi | A0141011J |

# Data Model



# 1. Rationale

Our data model rests on the following assumption(s) that we have made:

* Data duplication is acceptable
* Loss of information from original dataset is unacceptable
* Multiple reads (generally limited to 2) are preferred to retrieval and processing of entire table

Henceforth, we discuss each column family (and derived materialized view), in relation to the transaction listed, and justification for deviating from the given database schema. Note that attributes will be denoted in Courier. (Note also that due to space constraint, street\_1 and street\_2 has been condensed to street; however, in the actual implementation, street\_1 and street\_2).

## Transaction 1: New Order Transaction

The d\_next\_o\_id is read and incremented.

For each *OrderLine*, the *StockByWarehouse* is queried to retrieve the s\_quantity, i\_price, and i\_name in a single query. The rationale for joining the *Stock* table with the *Item* table allows for a single read query, instead of two read queries which may be more expensive especially for NUM\_ITEMS iteration.

The most popular item will be captured as the program iterates through the list of items, and will make note accordingly in the *Order* column family using attributes starting with popular\_item. Thereafter, a list of item IDs, ordered\_items, will be appended to the Order column family. The rationale is to reduce the number of reads for the other transactions. The *Order* row will then be created accordingly.

## Transaction 2: Payment Transaction

The *Warehouse, District and Customer* table are updated accordingly.

To reduce the number of read, Customer will contain the corresponding information (W\_STREET\_1, W\_STREET\_2, W\_CITY, W\_STATE, W\_ZIP) which is condensed in the JSON, w\_address; and (D\_STREET\_1, D\_STREET\_2, D\_CITY, D\_STATE, D\_ZIP) condensed in the JSON, d\_address.

## Transaction 3: Delivery Transaction

Each *District* for a given *Warehouse* shall be queried to get their respective last\_unfulfilled\_order, which serves as a tracker for the ID of the “oldest yet-to-be-delivered order”. After which, the same field will be incremented by one. The alternative is to query a subset of an *Order* column family, or *OrderLine* column family and perform processing to determine the “oldest yet-to-be-delivered order”, which may be prohibitively expensive depending on the cardinality of the data retrieved.

The relevant *Order, OrderLine, and Customer* rows (and columns) will then be updated accordingly once the ID is obtained.

Note that *Customer* will be queried, as certain information is required in the output. To reduce the read to *Warehouse*, and *District*, the customer’s row will contain the corresponding tax rate.

## Transaction 4: Order-Status Transaction

*Customer* is queried on his/her name and balance. Additionally, information pertaining to the last order, specifically, last\_order (O\_ID), last\_entry\_date (O\_ENTRY\_D), and last\_order\_carrier (O\_CARRIER\_ID) accompanies the *Customer* column family and will be queried accordingly. This should eliminate a separate call to query the *Order* column family.

Once the last\_order is obtained, the *OrderLine* column family will be queried.

## Transaction 5: Stock-level Transaction

*Order* will be queried on the last L order. The ordered\_itemsset will allow all the Orderline’s item ID to be retrieved.

With that, it becomes trivial to query *StockByWarehouse*, and output the required information.

## Transaction 6: Popular-Item Transaction

*Order* will be queried on the last L order. The popular item ID, name and quantity are associated with each *Order*.

Thereafter, the ordered\_itemsassociated with each *Order* allows for calculation to be made as to output “The percentage of orders… that contain the popular item” for each popular item.

## Transaction 7: Top-Balance Transaction

A materialised view, *CustomerByBalance* (derived from Customer) answers the query by ordering Customer table using c\_balance. To accommodate the requirement that w\_name and d\_name be output, the base column family (and by extension, the materialised view) shall contain the field.

# 2. Reference

Borsós, D. (2017, February 16). Everything you need to know about Cassandra Materialized Views. Retrieved September 09, 2017, from <https://opencredo.com/everything-need-know-cassandra-materialized-views/>

Carpenter, J., & Hewitt, E. (2016). *Cassandra: The Definitive Guide, 2nd Edition*. O'Reilly Media.

Data Types. (n.d.). Retrieved September 09, 2017, from <https://cassandra.apache.org/doc/latest/cql/types.html#grammar-token-collection_type>

Hobbs, T. (2017, May 04). Basic Rules of Cassandra Data Modeling. Retrieved September 10, 2017, from <https://www.datastax.com/dev/blog/basic-rules-of-cassandra-data-modeling>

Yeksigian, C. (2015, July 31). New in Cassandra 3.0: Materialized Views. Retrieved September 09, 2017, from <https://www.datastax.com/dev/blog/new-in-cassandra-3-0-materialized-views>

# 3. Performance Measure

The following are the benchmark derived from running…

## 3a. Database D8

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

## 3b. Database D40

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

# 4. Alternate Data Modelling

4a. Changes in Clustering / Partition Key Distribution in Compound Key

4b. Assumptions Questioned

# 5. Profiling for Changes in Parameter