CS4224: Distributed Database

Team 6

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

Data Model

<<CF>> WareHouse		
w_id	Int	K
w_name	Text	
w_street	Text	
w_city	Text	
w_state	Text	
w_zip	Text	
w_tax	Double	
w_ytd	Double	

<<CF>> OrderLine		
w_id	Int	K
d_id	Int	**C ↓**
o_id	Int	**C ↓**
ol_number	Int	**C ↑**
ol_i_id	Int	
ol_i_name	Text	
ol_delivery_d	Timestamp	
ol_amount	Double	
ol_supply_w_id	Int	
ol_quantity	Int	
ol_dist_info	Text	

<<CF>> StockByWarehouse		
w_id	Int	K
i_id	Int	C ↓
i_name	Text	
i_price	Double	
i_im_id	Int	
i_data	Text	
s_quantity	Int	
s_ytd	Int	
s_order_cnt	Int	++
s_remote_cnt	Int	++
s_dist_info	Text	
s_data	Text	

<<CF>> Customer		
w_id	Int	K
d_id	Int	**C ↓**
c_id	Int	**C ↓**
w_name	Text	
w_address	Json	
w_tax	Double	
d_name	Text	
d_address	Json	
d_tax	Double	
c_first	Text	
c_middle	Text	
c_last	Text	
c_street	Text	
c_city	Text	
c_state	Text	
c_zip	Text	
c_phone	Text	
c_since	Timestamp	
c_credit	Text	
c_credit_lim	Double	
c_discount	Double	
c_balance	Double	
c_ytd_payment	Double	
c_payment_cnt	Int	++
c_delivery_cnt	Int	++
c_data	Text	
last_order_id	Int	
last_order_date	Timestamp	
last_order_carrier	Int	

<<CF>> District		
w_id	Int	K
d_id	Int	C ↓
d_name	Text	
d_street	Text	
d_city	Text	
d_state	Text	
d_zip	Text	
d_tax	Double	
d_ytd	Double	
d_next_o_id	Int	++
last_unfulfilled_order	Int	++

<<CF>> Order		
w_id	Int	K
d_id	Int	**C ↓**
o_id	Int	**C ↓**
c_id	Int	
o_carrier_id	Int	
o_ol_cnt	Int	
o_all_local	Int	
o_entry_d	Timestamp	
c_first	Text	
c_middle	Text	
c_last	Text	
popular_item_id	Int	
popular_item_name	Text	
popular_item_qty	Int	
{ordered_items}	Set <Integer>	

<<MV>> CustomerByBalance		
w_id	Int	K
d_id	Int	**C ↓**
c_id	Int	**C ↓**
c_balance	Double	**C ↓**
w_name	Text	
d_name	Text	
c_first	Text	
c_middle	Text	
c_last	Text	

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`.

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_items` set 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_items` associated 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