National University of Singapore
School of Computing

CS4224/CS5424 Project
Semester 1, 2022/23

The objective of this project is to provide students the opportunity to acquire practical experience with using distributed database systems for application development. The learning tasks in this project involve the following: how to install a distributed database system on a cluster of machines, how to design a data model and implement transactions to support an application by taking into consideration of the workload of the application and the features of the database system, and how to benchmark the performance of an application.

In this project, you will be developing an application for a wholesale supplier using two different APIs of YugabyteDB (`https://www.yugabyte.com/`): a Cassandra-compatible API (YCQL `https://docs.yugabyte.com/preview/api/ycql/`) and a PostgreSQL-compatible API (YSQL `https://docs.yugabyte.com/preview/api/ysql/`).

Project team registration will be opened on August 16. Each team is to submit exactly one registration at LumiNUS > Survey > Project Team Registration by **August 23 (Tuesday, 11:59pm)**. A team with fewer than 5 members might be split or expanded with additional team member(s). A student not in any registered team will be assigned to a random team.

The project will be using the SoC Compute Cluster nodes. Each project team will be assigned to use a set of five cluster nodes. Details on the cluster assignment and how to use the cluster will be announced later.

1 Database Schema

The wholesale supplier has a number of geographically distributed sales districts and associated warehouses. Each regional warehouse covers 10 districts. Each district serves 3,000 customers. All warehouses maintain stocks for the 100,000 items sold by the supplier.

The relational schema for the wholesale supplier database consists of five entities (Warehouse, District, Customer, Order, Item) and two relationships (Order-Line, Stock). Order-Line specifies the items contained in each customer's order, and Stock specifies the availability information of items in each warehouse. The following subsections present the details of the schema. The key attribute(s) for each entity/relationship are shown in bold.

1.1 Warehouse

Attribute	Meaning	Type
W_ID	Warehouse number	INT
W_NAME	Warehouse name	VARCHAR(10)
W_STREET_1	Warehouse address	VARCHAR(20)
W_STREET_2	Warehouse address	VARCHAR(20)
W_CITY	Warehouse address	VARCHAR(20)
W_STATE	Warehouse address	CHAR(2)
W_ZIP	Warehouse address	CHAR(9)
W_TAX	Warehouse sales tax rate	DECIMAL(4,4)
W_YTD	Year to date amount paid to warehouse	DECIMAL(12,2)

1.2 District

Attribute	Meaning	Type
D_W_ID	Warehouse number	INT
D_ID	District number	INT
D_NAME	District name	VARCHAR(10)
D_STREET_1	District address	VARCHAR(20)
D_STREET_2	District address	VARCHAR(20)
D_CITY	District address	VARCHAR(20)
D_STATE	District address	CHAR(2)
D_ZIP	District address	CHAR(9)
D_TAX	District sales tax rate	DECIMAL(4,4)
D_YTD	Year to date amount paid to district	DECIMAL(12,2)
D_NEXT_O_ID	Next available order number for district	INT

D_W_ID is a foreign key that refers to Warehouse table.

1.3 Customer

Attribute	Meaning	Type
C_W_ID	Warehouse number	INT
C_D_ID	District number	INT
C_ID	Customer number	INT
C_FIRST	Customer name	VARCHAR(16)
C_MIDDLE	Customer name	CHAR(2)
C_LAST	Customer name	VARCHAR(16)
C_STREET_1	Customer address	VARCHAR(20)
C_STREET_2	Customer address	VARCHAR(20)
C_CITY,	Customer address	VARCHAR(20)
C_STATE	Customer address	CHAR(2)
C_ZIP	Customer address	CHAR(9)
C_PHONE	Customer phone	CHAR(16)
C_SINCE	Date and time when entry was created	TIMESTAMP
C_CREDIT	Customer credit status	CHAR(2)
C_CREDIT_LIM	Customer credit limit	DECIMAL(12,2)
C_DISCOUNT	Customer discount rate	DECIMAL(5,4)
C_BALANCE	Balance of customer's outstanding payment	DECIMAL(12,2)
C_YTD_PAYMENT	Year to date payment by customer	FLOAT
C_PAYMENT_CNT	Number of payments made	INT
C_DELIVERY_CNT	Number of deliveries made to customer	INT
C_DATA	Miscellaneous data	VARCHAR(500)

(C_W_ID, C_D_ID) is a foreign key that refers to District table.

1.4 Order

Attribute	Meaning	Type
O_W_ID	Warehouse number	INT
O_D_ID	District number	INT
O_ID	Order number	INT
O_C_ID	Customer number	INT
O_CARRIER_ID	Identifier of carrier who delivered the order	INT
O_OL_CNT	Number of items ordered	DECIMAL(2,0)
O_ALL_LOCAL	Order status (whether order includes only home order-lines)	DECIMAL(1,0)
O_ENTRY_D	Order entry data and time	TIMESTAMP

(O_W_ID, O_D_ID, O_C_ID) is a foreign key that refers to Customer table. The range of O_CARRIER_ID is [1,10].

1.5 Item

Attribute	Meaning	Type
I_ID	Item identifier	INT
I_NAME	Item name	VARCHAR(24)
I_PRICE	Item price	DECIMAL(5,2)
I_IM_ID	Item image identifier	INT
I_DATA	Brand information	VARCHAR(50)

1.6 Order-Line

Attribute	Meaning	Type
OL_W_ID	Warehouse number	INT
OL_D_ID	District number	INT
OL_O_ID	Order number	INT
OL_NUMBER	Order-line number	INT
OL_I_ID	Item number	INT
OL_DELIVERY_D	Data and time of delivery	TIMESTAMP
OL_AMOUNT	Total price for ordered item	DECIMAL(7,2)
OL_SUPPLY_W_ID	Supplying warehouse number	INT
OL_QUANTITY	Quantity ordered	DECIMAL(2,0)
OL_DIST_INFO	Miscellaneous data	CHAR(24)

(OL_W_ID, OL_D_ID, OL_O_ID) is a foreign key that refers to Order table. OL_I_ID is a foreign key that refers to Item table.

An order-line is classified as a home order-line if OL_SUPPLY_W_ID = OL_W_ID; otherwise, it it classified as a remote order-line. An order's O_ALL_LOCAL is set to *true* if and only if all its order-lines are home order-lines.

1.7 Stock

Attribute	Meaning	Type
S_W_ID	Warehouse number	INT
S_I_ID	Item number	INT
S_QUANTITY	Quantity in stock for item	DECIMAL(4,0)
S_YTD	Year to date total quantity ordered	DECIMAL(8,2)
S_ORDER_CNT	Number of orders	INT
S_REMOTE_CNT	Number of remote orders	INT
S_DIST_01	Information on district 1's stock	CHAR(24)
S_DIST_02	Information on district 2's stock	CHAR(24)
S_DIST_03	Information on district 3's stock	CHAR(24)
S_DIST_04	Information on district 4's stock	CHAR(24)
S_DIST_05	Information on district 5's stock	CHAR(24)
S_DIST_06	Information on district 6's stock	CHAR(24)
S_DIST_07	Information on district 7's stock	CHAR(24)
S_DIST_08	Information on district 8's stock	CHAR(24)
S_DIST_09	Information on district 9's stock	CHAR(24)
S_DIST_10	Information on district 10's stock	CHAR(24)
S_DATA	Miscellaneous data	VARCHAR(50)

S_I_ID is a foreign key that refers to Item table. S_W_ID is a foreign key that refers to Warehouse table.

2 Transactions

The workload for the wholesale supplier application consists of the following eight types of transactions:

1. **New Order Transaction** processes a new customer order.
2. **Payment Transaction** processes a customer payment for an order.
3. **Delivery Transaction** processes the delivery of the oldest yet-to-be-delivered order for each of the 10 districts in a specified warehouse.
4. **Order-Status Transaction** queries the status of the last order of a specified customer.
5. **Stock-Level Transaction** checks the stock level of items from a specified number of last orders at a warehouse district.
6. **Popular-Item Transaction** identifies the most popular items sold in each of a specified number of last orders at a specified warehouse district.
7. **Top-Balance Transaction** identifies the top-10 customers with the highest outstanding payment balance.
8. **Related-Customer Transaction** identifies the customers related to a specified customer.

The following subsections provide the specifications of these transaction types. The semantics of each transaction type is described in terms of a specific sequence of processing steps (you are free to implement the specification differently). For convenience, we use *null* to denote an unknown attribute value (you are free to choose an appropriate way to represent null values in your implementation). Of course, you are free to base your implementation on an entirely different schema design.

2.1 New Order Transaction

This transaction processes a new order from a customer.
Inputs:

1. Customer identifier (W_ID, D_ID, C_ID)
2. Number of items to be ordered NUM_ITEMS, NUM_ITEMS ≤ 20
3. ITEM_NUMBER[i] = Item number for i^{th} item, $i \in [1, NUM\_ITEMS]$
4. SUPPLIER_WAREHOUSE[i] = Supplier warehouse for i^{th} item, $i \in [1, NUM\_ITEMS]$
5. QUANTITY[i] = Quantity ordered for i^{th} item, $i \in [1, NUM\_ITEMS]$

Processing steps:

1. Let N denote value of the next available order number D_NEXT_O_ID for district (W_ID,D_ID)
2. Update the district (W_ID, D_ID) by incrementing D_NEXT_O_ID by one
3. Create a new order with
   - O_ID = N
   - O_D_ID = D_ID
   - O_W_ID = W_ID
   - O_C_ID = C_ID
   - O_ENTRY_D = Current date and time
   - O_CARRIER_ID = *null*
   - O_OL_CNT = NUM_ITEMS
   - O_ALL_LOCAL = 0 if there exists some $i \in [1, NUM\_ITEMS]$ such that SUPPLIER_WAREHOUSE[i] $\neq$ W_ID; otherwise, O_ALL_LOCAL = 1
4. Initialize TOTAL_AMOUNT = 0
5. For i = 1 to NUM_ITEMS
   (a) Let S_QUANTITY denote the stock quantity for item ITEM_NUMBER[i] and warehouse SUPPLIER_WAREHOUSE[i]
   (b) $ADJUSTED\_QTY = S\_QUANTITY - QUANTITY[i]$

(c) If ADJUSTED_QTY < 10, then set $ADJUSTED\_QTY = ADJUSTED\_QTY + 100$

(d) Update the stock for (ITEM_NUMBER[i], SUPPLIER_WAREHOUSE[i]) as follows:

- Update S_QUANTITY to ADJUSTED_QTY
- Increment S_YTD by QUANTITY[i]
- Increment S_ORDER_CNT by 1
- Increment S_REMOTE_CNT by 1 if SUPPLIER_WAREHOUSE[i] $\neq$ W_ID

(e) ITEM_AMOUNT = QUANTITY[i] $\times$ I_PRICE, where I_PRICE is the price of ITEM_NUMBER[i]

(f) TOTAL_AMOUNT = TOTAL_AMOUNT + ITEM_AMOUNT

(g) Create a new order-line

- OL_O_ID = N
- OL_D_ID = D_ID
- OL_W_ID = W_ID
- OL_NUMBER = i
- OL_I_ID = ITEM_NUMBER[i]
- OL_SUPPLY_W_ID = SUPPLIER_WAREHOUSE[i]
- OL_QUANTITY = QUANTITY[i]
- OL_AMOUNT = ITEM_AMOUNT
- OL_DELIVERY_D = *null*
- OL_DIST_INFO = S_DIST_xx, where xx = D_ID

6. TOTAL_AMOUNT = TOTAL_AMOUNT $\times (1 + D\_TAX + W\_TAX) \times (1 - C\_DISCOUNT)$, where W_TAX is the tax rate for warehouse W_ID, D_TAX is the tax rate for district (W_ID, D_ID), and C_DISCOUNT is the discount for customer C_ID.

Output the following information:

1. Customer identifier (W_ID, D_ID, C_ID), lastname C_LAST, credit C_CREDIT, discount C_DISCOUNT
2. Warehouse tax rate W_TAX, District tax rate D_TAX
3. Order number O_ID, entry date O_ENTRY_D
4. Number of items NUM_ITEMS, Total amount for order TOTAL_AMOUNT
5. For each ordered item ITEM_NUMBER[i], $i \in [1, NUM\_ITEMS]$
   (a) ITEM_NUMBER[i]
   (b) I_NAME
   (c) SUPPLIER_WAREHOUSE[i]
   (d) QUANTITY[i]
   (e) OL_AMOUNT
   (f) S_QUANTITY

2.2 Payment Transaction

This transaction processes a payment made by a customer.
Inputs:

1. Customer identifier (C_W_ID, C_D_ID, C_ID)
2. Payment amount PAYMENT

Processing steps:

1. Update the warehouse C_W_ID by incrementing W_YTD by PAYMENT
2. Update the district (C_W_ID,C_D_ID) by incrementing D_YTD by PAYMENT
3. Update the customer (C_W_ID, C_D_ID, C_ID) as follows:
   - Decrement C_BALANCE by PAYMENT
   - Increment C_YTD_PAYMENT by PAYMENT
   - Increment C_PAYMENT_CNT by 1

Output the following information:

1. Customer's identifier (C_W_ID, C_D_ID, C_ID), name (C_FIRST, C_MIDDLE, C_LAST), address (C_STREET_1, C_STREET_2, C_CITY, C_STATE, C_ZIP), C_PHONE, C_SINCE, C_CREDIT, C_CREDIT_LIM, C_DISCOUNT, C_BALANCE
2. Warehouse's address (W_STREET_1, W_STREET_2, W_CITY, W_STATE, W_ZIP)
3. District's address (D_STREET_1, D_STREET_2, D_CITY, D_STATE, D_ZIP)
4. Payment amount PAYMENT

2.3 Delivery Transaction

This transaction is used to process the delivery of the oldest yet-to-be-delivered order for each of the 10 districts in a specified warehouse.

Inputs:

1. Warehouse number W_ID
2. Identifier of carrier assigned for the delivery CARRIER_ID

Processing steps:

1. For DISTRICT_NO = 1 to 10
   (a) Let N denote the value of the smallest order number O_ID for district (W_ID,DISTRICT_NO) with O_CARRIER_ID = *null*; i.e.,

   $$N = \min\{t.O\_ID \in \text{Order} \mid t.O\_W\_ID = W\_ID,\ t.D\_ID = DISTRICT\_NO, t.O\_CARRIER\_ID = null\}$$

   Let X denote the order corresponding to order number N, and let C denote the customer who placed this order
   (b) Update the order X by setting O_CARRIER_ID to CARRIER_ID
   (c) Update all the order-lines in X by setting OL_DELIVERY_D to the current date and time
   (d) Update customer C as follows:
   - Increment C_BALANCE by B, where B denote the sum of OL_AMOUNT for all the items placed in order X
   - Increment C_DELIVERY_CNT by 1

2.4 Order-Status Transaction

This transaction queries the status of the last order of a customer.
Input: Customer identifier (C_W_ID, C_D_ID, C_ID)
Output the following information:

1. Customer's name (C_FIRST, C_MIDDLE, C_LAST), balance C_BALANCE
2. For the customer's last order
   (a) Order number O_ID
   (b) Entry date and time O_ENTRY_D
   (c) Carrier identifier O_CARRIER_ID
3. For each item in the customer's last order
   (a) Item number OL_I_ID
   (b) Supplying warehouse number OL_SUPPLY_W_ID
   (c) Quantity ordered OL_QUANTITY
   (d) Total price for ordered item OL_AMOUNT
   (e) Data and time of delivery OL_DELIVERY_D

2.5 Stock-level Transaction

This transaction examines the items from the last L orders at a specified warehouse district and reports the number of those items that have a stock level below a specified threshold.
Inputs:

1. Warehouse number W_ID
2. District number D_ID
3. Stock threshold T
4. Number of last orders to be examined L

Processing steps:

1. Let N denote the value of the next available order number D_NEXT_O_ID for district (W_ID,D_ID)
2. Let S denote the set of items from the last L orders for district (W_ID,D_ID); i.e.,

   $S = \{t.OL\_I\_ID \mid t \in \text{Order-Line},\ t.OL\_D\_ID = D\_ID,\ t.OL\_W\_ID = W\_ID,\ t.OL\_O\_ID \in [N-L, N)\}$

3. Output the total number of items in S where its stock quantity at W_ID is below the threshold; i.e., $S\_QUANTITY < T$

2.6 Popular-Item Transaction

This transaction finds the most popular item(s) in each of the last L orders at a specified warehouse district. Given two items X and Y in the same order O, X is defined to be more popular than Y in O if the quantity ordered for X in O is greater than the quantity ordered for Y in O.
Inputs:

1. Warehouse number W_ID
2. District number D_ID
3. Number of last orders to be examined L

Processing steps:

1. Let N denote the value of the next available order number D_NEXT_O_ID for district (W_ID,D_ID)
2. Let S denote the set of last L orders for district (W_ID,D_ID); i.e.,

$$S = \{t.O\_ID \mid t \in \text{Order},\ t.O\_D\_ID = D\_ID,\ t.O\_W\_ID = W\_ID,\ t.O\_ID \in [N - L, N)\}$$

3. For each order number x in S
   (a) Let I_x denote the set of order-lines for this order; i.e.,

$$I_x = \{t \in \text{Order-Line} \mid t.OL\_O\_ID = x,\ t.OL\_D\_ID = D\_ID,\ t.OL\_W\_ID = W\_ID\}$$

   (b) Let $P_x \subseteq I_x$ denote the subset of popular items in I_x; i.e.,

$$t \in P_x \iff \forall\ t' \in I_x,\ t'.OL\_QUANTITY \leq t.OL\_QUANTITY$$

Output the following information:

1. District identifier (W_ID, D_ID)
2. Number of last orders to be examined L
3. For each order number x in S
   (a) Order number O_ID & entry date and time O_ENTRY_D
   (b) Name of customer who placed this order (C_FIRST, C_MIDDLE, C_LAST),
   (c) For each popular item in P_x
      i. Item name I_NAME
      ii. Quantity ordered OL_QUANTITY
4. The percentage of examined orders that contain each popular item
   (a) For each distinct popular item $t \in P_x,\ x \in S$
      i. Item name I_NAME
      ii. The percentage of orders in S that contain the popular item t

2.7 Top-Balance Transaction

This transaction finds the top-10 customers ranked in descending order of their outstanding balance payments.

Processing steps:

1. Let $C \subseteq$ Customer denote the subset of 10 customers (i.e., $|C| = 10$) such that for each pair of customers (x, y), where $x \in C$ and $y \in$ Customer $- C$, we have $x.C\_BALANCE \geq y.C\_BALANCE$.

Output the following information:

1. For each customer in C ranked in descending order of C_BALANCE:
   (a) Name of customer (C_FIRST, C_MIDDLE, C_LAST)
   (b) Balance of customer's outstanding payment C_BALANCE
   (c) Warehouse name of customer W_NAME
   (d) District name of customer D_NAME

2.8 Related-Customer Transaction

This transaction finds all the customers who are related to a specific customer. Given a customer C, another customer C' is defined to be related to C if all the following conditions hold:

- C and C' are associated with different warehouses; i.e., $C.C\_W\_ID \neq C'.C\_W\_ID$, and
- C and C' each has placed some order, O and O', respectively, where both O and O' contain at least two items in common.

Input: Customer identifier (C_W_ID, C_D_ID, C_ID)
Processing steps:

1. Let S be the set of customers who are related to the customer identified by (C_W_ID, C_D_ID, C_ID).

$$S = \{C' \in Customer \mid C'.C\_W\_ID \neq C\_W\_ID,$$
$$\exists\ O \in Order,\ O.O\_W\_ID = C\_W\_ID,\ O.O\_D\_ID = C\_D\_ID,\ O.O\_C\_ID = C\_ID,$$
$$\exists\ O' \in Order, O'.O\_W\_ID = C'.C\_W\_ID,\ O'.O\_D\_ID = C'.C\_D\_ID,\ O'.O\_C\_ID = C'.C\_ID$$
$$\exists\, OL1 \in OrderItem,\ OL1.OL\_W\_ID = O.O\_W\_ID,\ OL1.OL\_D\_ID = O.O\_D\_ID,\ OL1.OL\_O\_ID = O.O\_ID,$$
$$\exists\, OL2 \in OrderItem,\ OL2.OL\_W\_ID = O.O\_W\_ID,\ OL2.OL\_D\_ID = O.O\_D\_ID,\ OL2.OL\_O\_ID = O.O\_ID,$$
$$\exists\, OL1' \in OrderItem,\ OL1'.OL\_W\_ID = O'.O\_W\_ID,\ OL1'.OL\_D\_ID = O'.O\_D\_ID,\ OL1'.OL\_O\_ID = O'.O\_ID,$$
$$\exists\, OL2' \in OrderItem,\ OL2'.OL\_W\_ID = O'.O\_W\_ID,\ OL2'.OL\_D\_ID = O'.O\_D\_ID,\ OL2'.OL\_O\_ID = O'.O\_ID,$$
$$OL1.OL\_I\_ID \neq OL2.OL\_I\_ID,\ OL1'.OL\_I\_ID \neq OL2'.OL\_I\_ID,$$
$$OL1.OL\_I\_ID = OL1'.OL\_I\_ID,\ OL2.OL\_I\_ID = OL2'.OL\_I\_ID\}$$

Output the following information:

1. Customer identifier (C_W_ID, C_D_ID, C_ID)
2. For each customer $C \in S$
   (a) Output the identifier of C

2.9 Transaction Workload

In this project, we consider a transaction workload with the following distribution for the eight transaction types:

Transaction Type	**Frequency (%)**
New-Order	40
Payment	20
Delivery	20
Order-Status	4
Stock-Level	4
Popular-Item	4
Top-Balance	6
Related-Customer	2

The transactions satisfy the following properties:

1. Number of items per new order: $[4, 20]$
2. Stock threshold T in Stock-level transactions: $[1, 10]$
3. Number L in Stock-level and Popular-item transactions: $[20, 50]$

3 What to do

The project consists of two parts with each part comprising of four main tasks. The first part is to be done using **YugabyteDB's YSQL API**, and the second part is to be done using **YugabyteDB's YCQL API**. You will use **YugabyteDB's version 2.14.1.0 (build 2.14.1.0-b36)** for the project. The project deliverables are described in Section 5. The maximum score for each part is 15 marks.

For each part (using a specific API of YugabyteDB), the goal is to make the best use of the features provided by YugabyteDB to design and develop an efficient approach to process the transaction workload. Specifically, for each YugabyteDB API, there are four tasks:

1. **Data Modeling.** The first task is to design a data model to support the application's transaction workload.

2. **Transaction Implementation.** The second task is to implement the application with the following nine functions/programs:

   (a) A program/script for loading data into your database.

   (b) One function for each of the eight transaction types.

   (c) A main driver program that simulates a client executing transactions on the database. This driver function reads its inputs from `stdin`, where each input specifies an instance of one of the eight transaction types. For each transaction read, the driver invokes the appropriate transaction function to execute that transaction. The driver repeatedly reads a transaction from `stdin` and processes that transaction until EOF is read. The driver function must process the transactions sequentially; i.e., the current transaction must be completed before reading and processing the next transaction. Any output to be produced by a transaction should be written into `stdout`. At the end of processing all the transactions, the driver outputs to `stderr` the following information:

   - Total number of transactions processed
   - Total elapsed time for processing the transactions (in seconds)
   - Transaction throughput (number of transactions processed per second)
   - Average transaction latency (in ms)
   - Median transaction latency (in ms)
   - 95th percentile transaction latency (in ms)
   - 99th percentile transaction latency (in ms)

   The input specification for each transaction type consists of at least one line with each line consisting of comma-separated parameter values. The first value in the first line must be either N, P, D, O, S, I, T, or R, denoting, respectively, a new order, payment, delivery, order-status, stock-level, popular-item, top-balance, or related-customer transaction. The input specification for each of the transaction types is defined as follows:

   (a) **New Order Transaction** consists of M+1 lines, where M denote the number of items in the new order. The first line consists of five comma-separated values: N,C_ID,W_ID,D_ID,M.

Each of the M remaining lines specifies an item in the order and consists of three comma-separated values: OL_I_ID,OL_SUPPLY_W_ID,OL_QUANTITY.

(b) **Payment Transaction** consists of one line of input with five comma-separated values: P,C_W_ID,C_D_ID,C_ID,PAYMENT.

(c) **Delivery Transaction** consists of one line of input with three comma-separated values: D,W_ID,CARRIER_ID.

(d) **Order-Status Transaction** consists of one line of input with four comma-separated values: O,C_W_ID,C_D_ID,C_ID.

(e) **Stock-Level Transaction** consists of one line of input with five comma-separated values: S,W_ID,D_ID,T,L.

(f) **Popular-Item Transaction** consists of one line of input with four comma-separated values: I,W_ID,D_ID,L.

(g) **Top-Balance Transaction** consists of one line of input with one value: T.

(h) **Related-Customer Transaction** consists of one line of input with four comma-separated values: R,C_W_ID,C_D_ID,C_ID.

3. **Installation.** The third task is to install and configure YugabyteDB on a cluster of 5 servers. The database is to be partitoned among 5 servers using a replication factor of 3 (i.e., there are three copies of each database record). Your database should be installed in each node's local disk at `/temp` (not to be confused with /tmp). An initial set of data will be provided to you to initialize your database with data.

4. **Performance Measurement.** The fourth task is to measure the performance of your application with 20 clients $(C_0, C_1, \cdots, C_{19})$ executing transactions concurrently from the cluster of 5 servers $(S_0, S_1, S_2, S_3, S_4)$ using a set of 20 transaction specification files $(0.txt, 1.txt, \cdots, 19.txt)$. The transaction specification files will be provided to you. Each client C_i is simulated by executing an instance of your main driver program on server $S_{(i \ mod \ 5)}$ which reads from the transaction specification file named `i.txt`. Thus, there are 4 clients executing transactions concurrently on each server.

   Report the following performance measurements:

   (1) As stated in task 2(c), on completion of its transaction executions, each client reports the following performance measurements (with all floating point measurement values rounded off to two decimal places):

   (a) Number of executed transactions

   (b) Total transaction execution time (in seconds)

   (c) Transaction throughput (i.e., number of executed transactions per second given by (a) / (b))

   (d) Average transaction latency (in ms)

   (e) Median transaction latency (in ms)

(f) 95th percentile transaction latency (in ms)

(g) 99th percentile transaction latency (in ms)

Record these 7 measurements in a single 8-column CSV file named **clients.csv**, with the following schema for each line:

client_number, measurement_a, $\cdots$, measurement_g

This file should have exactly 20 lines.

(2) The minimum, average, and maximum transaction throughputs among the 20 clients. Record these 3 measurements in a single 3-column CSV file named **throughput.csv**, with the following schema for each line:

min_throughput, max_throughput, avg_throughput

(3) Report the database state at the end of the performance measurement by computing the following 15 statistics from the database; for convenience, we specify the semantics of these 15 statistics using the following 6 SQL queries:

i. select sum(W_YTD) from Warehouse

ii. select sum(D_YTD), sum(D_NEXT_O_ID) from District

iii. select sum(C_BALANCE), sum(C_YTD_PAYMENT), sum(C_PAYMENT_CNT), sum(C_DELIVERY_CNT) from Customer

iv. select max(O_ID), sum(O_OL_CNT) from Order

v. select sum(OL_AMOUNT), sum(OL_QUANTITY) from Order-Line

vi. select sum(S_QUANTITY), sum(S_YTD), sum(S_ORDER_CNT), sum(S_REMOTE_CNT) from Stock

Record these 15 statistics in a 15-line file named **dbstate.csv**, with one statistic on one line:

sum(W_YTD)
sum(D_YTD)
sum(D_NEXT_O_ID)
sum(C_BALANCE)
sum(C_YTD_PAYMENT)
sum(C_PAYMENT_CNT)
sum(C_DELIVERY_CNT)
max(O_ID)
sum(O_OL_CNT)
sum(OL_AMOUNT)
sum(OL_QUANTITY)
sum(S_QUANTITY)
sum(S_YTD)
sum(S_ORDER_CNT)
sum(S_REMOTE_CNT)

As the performance of these statistic computations will not be measured, your database schema design should be optimized for the performance of only the transaction executions (and not for these statistics computations).

4 Files to Download

The project's data and transaction specification files (size: 382MB) can be downloaded from

`http://www.comp.nus.edu.sg/~cs4224/project_files.zip`

The unzipped directory **project_files** consists of two sub-directories:

- **data_files/** consists of 7 CSV files representing the data for the 5 entities and 2 relationships in the database schema: customer.csv, district.csv, item.csv, order.csv, order-line.csv, stock.csv, and warehouse.csv. The database should be initialized using the data from these files for the performance measurement.
- **xact_files/** consists of 20 transaction specification files (0.txt to 19.txt). Each file **i.txt** represents the sequence of transactions issued by client i. These files are to be used for simulating client requests in the performance measurement.

5 Project Deliverables

The project consists of two deliverables with the following submission deadlines:

Deliverable	Deadline
One-page progress report	October 7 (Friday), 23:59
Project report & code submission	November 4 (Friday), 23:59

5.1 Progress Report

1. A progress report (must be in pdf format, maximum of 1 page, minimum font size 10). The progress report should include the following:
   (a) Team identifier and names of all team members.
   (b) For each team member, list of his/her allocated tasks.
   (c) Progress for YSQL implementation
      i. A list of completed tasks.
      ii. A list of in-progress tasks.
   (d) Progress for YCQL implementation
      i. A list of completed tasks.
      ii. A list of in-progress tasks.
   (e) Any other comments (optional).

5.2 Final Report & Code

1. A report (must be in pdf format, maximum of 30 pages, minimum font size 10). The report should include the following:
   (a) Team identifier and names of all team members on the first page.
   (b) For each team member, state his/her contributions to the project.
   (c) For each part (YSQL and YCQL implementation)
      i. Description of the data models with justifications for the design decisions.
      ii. A brief description of your implementation of the transaction functions. You should not repeat the contents of the transaction specifications here; instead, focus on explaining why your implementation is efficient, etc.
      iii. A brief discussion of the essential configuration options used
      iv. A discussion of the results for your performance benchmarking.
   (d) A discussion of the performance of YSQL vs YCQL, any lessons learnt, difficulties encountered, etc.
   (e) A list of all references (e.g., books, papers, websites) consulted for the project.
2. For each part (YSQL and YCQL implementation)
   (a) Source code of your application (including all scripts/configuration files). Marks will be penalized for incomplete submissions.
   (b) A detailed README file describing how to configure, setup and run your application using your submitted code/configuration files.
   (c) The files **clients.csv**, **throughput.csv**, and **dbstate.csv**.

5.3 How to Submit

Each team should submit four files in total; do not make multiple submissions of each file.

Upload your project deliverables into the following four **LumniNUS** > **Files** folders as follows:

- **Project progress report**: Name your file as X_progress.pdf
- **Project final report**: Name your file as X_final.pdf
- **YSQL source code**: Name your file as X_ysql.zip
- **YCQL source code**: Name your file as X_ycql.zip

In the above file names, "X" denote your team identifier (i.e., $a, \cdots, z, aa, \cdots, ae$).

Late submission policy: Any submission that is late by one day will incur a penalty of 2 marks; any submission that is late by two days will incur a penalty of 4 marks; and any submission that is late by more than two days will not be graded and will receive 0 marks. Late days are rounded up to the nearest integer. For example, a submission that is late by 1 hour will count as 1 day late. So if your Cassandra code submission is late by 5 hours and your final report is late by 30 hours, your total penalty will be 6 marks.

6 Project Evaluation

1. Your implementation may be tested on a different cluster of machines. It is important that your submitted README file clearly explains how to configure and run your implementation, and how to compile and generate any required executable files.

2. It is important that your implementation can be executed using the command line interface without the need to install any IDE environment.

3. We will not be making any modifications to your source code. It is therefore important that all parameter configurations (e.g., IP address) be specified in scripts instead of in source code.

4. Your project will be graded based on the following criteria: data model design, implementation of transactions, experimental evaluations, and written report.

5. There will a 20-minute project evaluation session for each team during week 13. Each evaluation session involves a combination of Q&A and demo of implemented code. All team members are expected to attend the evaluation session. The registration of the evaluation sessions will be announced in week 12.