TPC-C DATABASE TRANSACTIONS

June 2019

Contents

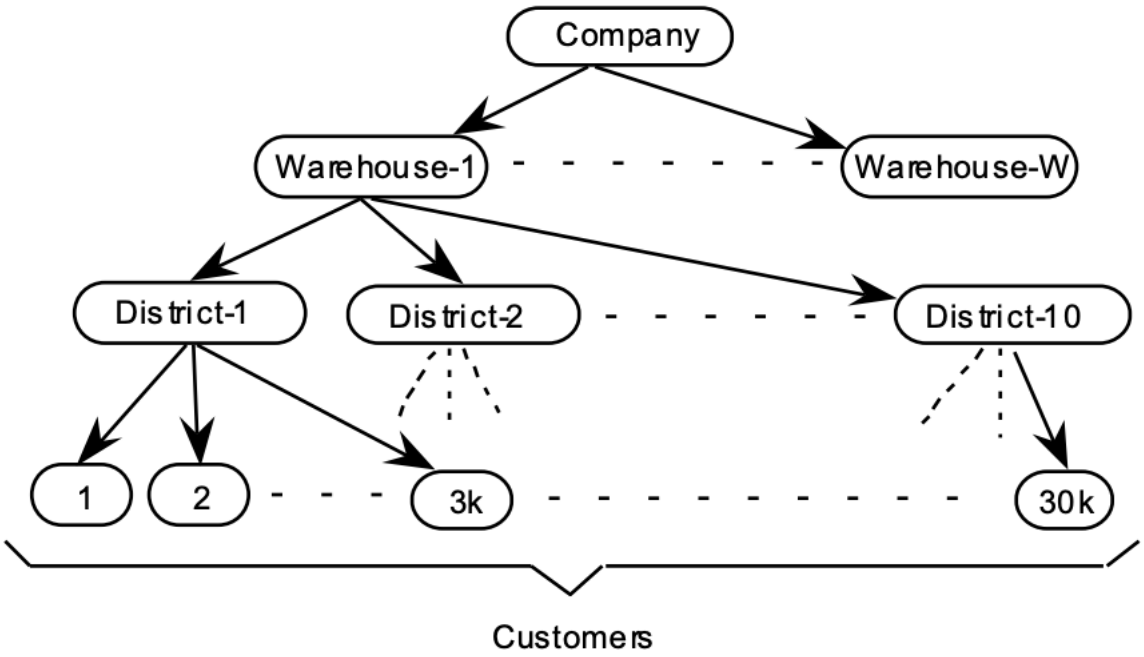
1 TPC-C Benchmark Guide

1.1 Model summary

Understanding of the database and the structure of the Company is essential to understanding the code.

As described in Clause 1 (page 10):

> The Company portrayed by the benchmark is a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. As the Company's business expands, new warehouses and associated sales districts are created. Each regional warehouse covers 10 districts. Each district serves 3,000 customers. All warehouses maintain stocks for the 100,000 items sold by the Company. The following diagram illustrates the warehouse, district, and customer hierarchy of TPC-C's business environment.



> Customers call the Company to place a new order or request the status of an existing order. Orders are composed of an average of 10 order lines (i.e. line items). One percent of all order lines are for items not in-stock at the regional warehouse and must be supplied by another warehouse.

The Company's system is also used to enter payments from customers, process orders for delivery, and examine stock levels to identify potential supply shortages.

1.2 Abridged table chema

The TPC-C database contains nine tables. Primary keys are bolded and underlined below.

WAREHOUSE(**wId**, name, str1, str2, city, state, zip, tax, wYTD)

DISTRICT(**dId, wId***, name, str1, str2, city, state, zip, tax, dYTD, nextOrdId)

CUSTOMER(**cId, dId, wId**, first, middle, last, str1, str2, city, state, zip, phone, since, credit, creditLim, discount, balance, YTDPaymt, paymtCnt, deliveryCnt, data)

HISTORY(cId, cDId, cWId, dId, wId, date, amount, data)

NEWORDER(**oId, dId, wId**)

ORDER(**oId, dId*, wId***, cId*, entryDate, carrierId, oLCnt, allLocal)

ORDERLINE(**oId, dId*, wId*, number**, iId, supplyWId, deliveryDate, quantity, amount, distInfo)

ITEM(**iId**, iMId, name, price, data)

STOCK(**iId*, wId***, quantity, dist01, dist02, dist03, dist04, dist05, dist06, dist07, dist08, dist09, dist10, YTD, orderCnt, remoteCnt, data)

Foreign keys:

DISTRICT(wId) references WAREHOUSE(wId)

CUSTOMER(wId, dId) references DISTRICT(wId, dId)

HISTORY(cWId, cDId, cId) references CUSTOMER(wId, dId, cId)

HISTORY (wId, dId) references DISTRICT(wId, dId)

NEWORDER(wId, dId, oId) references ORDER(wId, dId, oId)

ORDER (wId, dId, cId) references CUSTOMER(wId, dId, cId)

ORDERLINE(wId, dId, oId) references ORDER(wId, dId, oId)

ORDERLINE(supplyWId, iId) references STOCK(wId, iId)

STOCK(wId) references WAREHOUSE(wId)

STOCK(iId) references ITEM(iId)

See Annex 1 for table cardinalities and layouts.

2 Database generation

```
DROP TABLE IF EXISTS WAREHOUSE CASCADE;
DROP TABLE IF EXISTS DISTRICT CASCADE;
DROP TABLE IF EXISTS CUSTOMER CASCADE;
DROP TABLE IF EXISTS HISTORY CASCADE;
DROP TABLE IF EXISTS NEW_ORDER CASCADE;
DROP TABLE IF EXISTS ORDERS CASCADE;
DROP TABLE IF EXISTS ORDERLINE CASCADE;
DROP TABLE IF EXISTS ITEM CASCADE;
DROP TABLE IF EXISTS STOCK CASCADE;


CREATE TABLE WAREHOUSE(
  wId INTEGER PRIMARY KEY,
  name VARCHAR(10),
  str1 VARCHAR(20),
  str2 VARCHAR(20),
  city VARCHAR(20),
  state CHAR(2),
  zip CHAR(9),
  tax NUMERIC(4,4),
  wYTD NUMERIC(12,2)
);

CREATE TABLE DISTRICT(
  dId INTEGER UNIQUE,
  wId INTEGER UNIQUE,
  name VARCHAR(10),
  str1 VARCHAR(20),
  str2 VARCHAR(20),
  city VARCHAR(20),
  state CHAR(2),
  zip CHAR(9),
  tax NUMERIC(4,4),
  dYTD NUMERIC(12,2),
```

```
  nextOrdId INTEGER UNIQUE,
  PRIMARY KEY(dId, wId),
  FOREIGN KEY(wId) REFERENCES WAREHOUSE(wId)
);

CREATE TABLE CUSTOMER(
  cId INTEGER UNIQUE,
  dId INTEGER UNIQUE,
  wId INTEGER UNIQUE,
  first VARCHAR(16),
  middle CHAR(2),
  lasT VARCHAR(16),
  str1 VARCHAR(20),
  str2 VARCHAR(20),
  city VARCHAR(20),
  state CHAR(2),
  zip CHAR(9),
  phone CHAR(16),
  since DATE,
  credit CHAR(2),
  creditLim NUMERIC(12,2),
  discount NUMERIC(4,4),
  balance NUMERIC(12,2),
  YTDPaymt NUMERIC(12,2),
  paymtCnt NUMERIC(4),
  deliveryCnt NUMERIC(4),
  data VARCHAR(50),
  PRIMARY KEY(cId,dId, wId)
);

CREATE TABLE HISTORY(
  cId INTEGER UNIQUE,
  cDId INTEGER UNIQUE,
  cWId INTEGER UNIQUE,
  dId INTEGER UNIQUE,
```

```
  wId INTEGER UNIQUE,
  date date,
  amount NUMERIC(6,2),
  data VARCHAR(24),
  FOREIGN KEY (cWId, cDId, cId) REFERENCES CUSTOMER(wId, dId, cId),
  FOREIGN KEY (wId,dId) REFERENCES DISTRICT(wId,dId)
);

CREATE TABLE ORDERS(
  oId INTEGER UNIQUE,
  dId INTEGER UNIQUE,
  wId INTEGER UNIQUE,
  cId INTEGER UNIQUE,
  entryDate DATE,
  carrierId INTEGER,
  oLCnt NUMERIC(2),
  allLocal NUMERIC(1),
  PRIMARY KEY (oId,dId,wId),
  FOREIGN KEY (wId, dId, cId) REFERENCES CUSTOMER(wId, dId, cId)
);

CREATE TABLE NEW_ORDER(
  oId INTEGER PRIMARY KEY,
  dId INTEGER UNIQUE,
  wId INTEGER UNIQUE,
  FOREIGN KEY (wId,dId,oId) REFERENCES ORDERS(wId,dId,oId)
);



CREATE TABLE ITEM(
  iId INTEGER PRIMARY KEY,
  iMId INTEGER UNIQUE,
  name VARCHAR(24),
  price NUMERIC(5,2),
  data VARCHAR(50)
```

```
);

CREATE TABLE STOCK(
  iId INTEGER UNIQUE,
  wId INTEGER UNIQUE,
  quantity NUMERIC(4),
  dist01 CHAR(24),
  dist02 CHAR(24),
  dist03 CHAR(24),
  dist04 CHAR(24),
  dist05 CHAR(24),
  dist06 CHAR(24),
  dist07 CHAR(24),
  dist08 CHAR(24),
  dist09 CHAR(24),
  dist10 CHAR(24),
  YTD NUMERIC(8),
  orderCnt NUMERIC(4),
  remoteCnt NUMERIC(4),
  data VARCHAR(50),
  PRIMARY KEY (iId,wId),
  FOREIGN KEY (wId) REFERENCES WAREHOUSE(wId),
  FOREIGN KEY (iId) references ITEM(iId)
);

CREATE TABLE ORDERLINE(
  oId INTEGER UNIQUE,
  dId INTEGER UNIQUE,
  wId INTEGER UNIQUE,
  number INTEGER UNIQUE,
  iId INTEGER UNIQUE,
  supplyWId INTEGER UNIQUE,
  deliveryDate DATE,
  quantity NUMERIC(2),
  amount NUMERIC(6,2),
```

```
  distInfo CHAR(24),
  PRIMARY KEY (oId, dId, wId, number),
  FOREIGN KEY (wId, dId, oId) REFERENCES ORDERS(wId, dId, oId),
  FOREIGN KEY (supplyWId, iId) REFERENCES STOCK(wId, iId)
);
```

3 PL/pgSQL

PL/pgSQL is a loadable procedural language for the PostgreSQL database system. See documentation here: https://www.postgresql.org/docs/10/plpgsql-overview.html

Some notes about the PL/pgSQL syntax with regards to the functions presented in this report:

- All variables referred to in each function have to either be among the parameters of the function, or declared at the beginning of the function.

- The return type of a function must be specified at the beginning of the function. A function that returns nothing should specify with RETURNS VOID. All of the functions below return nothing or 0, but could be modified to return desired values.

4 TPC-C functions

The TPC-C document presents 5 transactions: new order (create new order), payment, order status, delivery, and stock level. The document also presents the code for a sample load program.

4.1 The New-Order Transaction

4.1.1 Overview

The neworder function enters a complete order through a single database transaction. During the transaction, a new order is created and added into the database, the total amount to pay is calculated, and the quantity of items in stock is changed accordingly.

4.1.2 The neworder Function

```
CREATE OR REPLACE FUNCTION neworder (w_Id INTEGER,d_id INTEGER,
c_Id INTEGER, ol_cnt NUMERIC(2), ol_i_id INTEGER, ol_supply_w_id,
```

```
ol_quantity NUMERIC(2))

RETURNS VOID AS $$

DECLARE
  c_discount NUMERIC(4,4);
  c_last VARCHAR(16);
  c_credit CHAR(2);
  w_tax NUMERIC(4,4);
  d_next_o_id INTEGER;
  d_tax NUMERIC(4,4),
  i_price NUMERIC(5,2);
  i_name VARCHAR(24);
  i_data VARCHAR(50);
  o_id INTEGER;
  ol_amount NUMERIC(4,4);

BEGIN
  SELECT C.discount, C.last, C.credit, W.tax
  INTO c_discount, c_last, c_credit, w_tax
  FROM CUSTOMER C, WAREHOUSE W
  WHERE W.wId = w_id AND C.wId = W.wId AND C.dId = d_id AND
  C.cId = c_Id;

  SELECT nextOrdId, tax
  INTO d_next_o_id, d_tax
  FROM DISTRICT D
  WHERE D.dId = d_id AND D.wId = w_id;

  UPDATE DISTRICT SET nextOrdId = d_next_o_id + 1
  WHERE DISTRICT.dId = d_id AND DISTRICT.wId = w_id;

  o_id := d_next_o_id;

  INSERT INTO ORDERS(oId, dId, wId, cId, entryDate, oLCnt, allLocal)
```

```
VALUES (o_id, d_Id, w_Id, c_Id, date_time, o_ol_cnt, o_all_local);

INSERT INTO NEW_ORDER(oId, dId, wId)
VALUES (o_id, d_Id, w_Id);

FOR ol_number in 1 .. o_ol_cnt LOOP
  ol_supply_w_id := supplywarehouse[ol_number]; -- modified from
     original code since Postgres array index starts from 1
  -- get supply warehouse id
  IF (ol_supply_w_id != w_id) THEN
    o_all_local := 0;
    -- if supplying warehouse != home warehouse, then not local
  END IF;
  ol_i_id := itemid[ol_number];
  ol_quantity := qty[ol_number];

  SELECT I.price, I.name, I.data
  INTO i_price, i_name, i_data
  FROM ITEM
  WHERE ITEM.iId = ol_i_id;

  price[ol_number] := i_price;
  iname[ol_number] := i_name;

  SELECT quantity, data, dist01, dist02, dist03, dist04, dist05,
  dist06, dist07, dist08, dist09, dist10
  INTO s_quantity, s_data, s_dist01, s_dist02, s_dist03, s_dist04,
  s_dist05, s_dist06, s_dist07, s_dist08, s_dist09, s_dist10
  FROM STOCK
  WHERE STOCK.iId = ol_i_id AND STOCK.wId = ol_supply_w_id;

  -- pick correct s_dist_xx
  pick_dist_info(ol_dist_info, ol_w_id);
  stock[ol_number - 1] := s_quantity;
```

```
    IF (i_data LIKE '%original' AND s_data LIKE '%original') THEN
      bg[ol_number] := 'B';
    ELSE
      bg[ol_number] := 'G';
    END IF;

    IF (s_quantity > ol_quantity) THEN
      s_quantity := s_quantity - ol_quantity;
    ELSE
      s_quantity := s_quantity - ol_quantity + 91;
    END IF;

    UPDATE STOCK SET quantity = s_quantity
    WHERE STOCK.iId = ol_i_id
    AND STOCK.wId = ol_supply_w_id;

    ol_amount := ol_quantity * i_price * (1 + w_tax + d_tax) *
    (1 - c_discount);
    amt[ol_number] := ol_amount;
    total :+= ol_amount;

    INSERT INTO ORDERLINE(oId, dId, wId, number, iId, supplyWId,
    quantity, amount, distInfo)
    VALUES (o_id, d_id, w_id, ol_number, ol_i_id, ol_supply_w_id,
    ol_quantity, ol_amount, ol_dist_info)
  END LOOP;
END;

$$ LANGUAGE PLpgSQL;
```

4.1.3 The Transaction Profile

For a given warehouse number w_id, district number d_id, customer number c_id, count of items (ol_cnt), and for a given set of items (ol_i_id), supplying warehouses (ol_supply_w_id), and quantities (ol_quantity):

1. The row in the DISTRICT (D) table with matching D.wId and D.dId is selected, D.tax (the district tax rate) is retrieved.

2. D.nextOrdId (the next available order number for the district) is retrieved and incremented by 1.

3. A new row is inserted into both the ORDERS (O) and NEW_ORDER (NO) table to reflect the creation of the new order.

4. **For each item on the order (comprised of o_ol_cnt items):**

   (a) The supply warehouse ID is selected from list of warehouses. If the supply warehouse ID is same as the home warehouse ID then O.allLocal is set to 1, otherwise O.allLocal is set to 0. The corresponding values for ol_i_id and ol_quantity is selected from the itemid and qty arrays.

   (b) The row in the ITEM (I) table with matching I.iId (i.e. equals ol_i_id) is selected and I.price (the price of the item), I.name (the name of the item) and I.data are retrieved.

   (c) The row in the STOCK (S) table with matching S.iId (i.e. equals ol_i_id) is selected and S.wId (i.e. equals ol_supply_w_id) is selected. S.quantity (the quantity in stock), S.distxx (where xx represents the district number), and S.data are retrieved. The stock of the item is updated in the stock array.

   (d) The strings in I.data and S.data are examined. If they both include the string "ORIGINAL", the brand-generic field for that item is set to "B", otherwise, the brand-generic field is set to "G".

   (e) If the retrieved value for s_quantity exceeds ol_quantity then s_quantity is decreased by ol_quantity, otherwise s_quantity is updated to (s_quantity - ol_quantity) + 91 (i.e. if not enough items are in stock to fulfill this order, then the item is restocked by 91 units before being subtracted by the quantity of items being purchased, otherwise the stock quantity of the item is subtracted by the quantity being purchased).

   (f) ol_amount (the amount for the item in the order) is computed as: ol_quantity * i_price. Adding all the items together, the total amount for the order is computed as: SUM(ol_amount) * (1 - c_discount) * (1 + w_tax + d_tax).

   (g) A new row is inserted into the ORDERLINE table to reflect the item on the order.

4.2 The Payment Transaction

4.2.1 Overview

The payment business transaction updates the customer's balance and reflects the payment on the district and warehouse sales statistics. A customer can be selected by their last name or by their unique customer ID.

A Payment transaction is said to be **home** if the customer belongs to the warehouse from which the payment is entered (when CUSTOMER.wId = w_id)

A Payment transaction is said to be **remote**if the warehouse from which the payment is entered is not the one to which the customer belongs to (when CUSTOMER.wId does not equal w_id.

4.2.2 The payment Function

```
CREATE OR REPLACE FUNCTION payment(w_id INTEGER, d_id INTEGER, c_id
    INTEGER, c_last VARCHAR(16),h_amount NUMERIC(12,2),by_name BOOLEAN)

RETURNS INTEGER AS $$

DECLARE
  w_str1 VARCHAR(20);
  w_str2 VARCHAR(20);
  w_city VARCHAR(20);
  w_state CHAR(2);
  w_zip CHAR(9);
  w_name VARCHAR(10);
  d_str1 VARCHAR(20);
  d_str2 VARCHAR(20);
  d_city VARCHAR(20);
  d_state CHAR(2);
  d_zip CHAR(9);
  d_name VARCHAR(10);
  name_cnt INTEGER;
  c_first VARCHAR(16);
  c_middle CHAR(2);
  c_id INTEGER;
```

```
  c_str1 VARCHAR(20);
  c_str2 VARCHAR(20);
  c_city VARCHAR(20);
  c_state CHAR(2);
  c_zip CHAR(9);
  c_phone CHAR(16);
  c_credit CHAR(2);
  c_creditLim NUMERIC(12,2);
  c_discount NUMERIC(4,4);
  c_balance NUMERIC(12,2);
  c_since DATE;
  n INTEGER := 0;
BEGIN
  UPDATE WAREHOUSE SET wYTD = wYTD + h_amount
  WHERE WAREHOUSE.wId = w_id;

  SELECT str1, str2, city, state, zip, name
  INTO w_str1, w_str2, w_city, w_state, w_zip, w_name
  FROM WAREHOUSE W
  WHERE W.wId = w_id;

  UPDATE DISTRICT SET dYTD = dYTD + h_amount
  WHERE DISTRICT.wId = w_id AND DISTRICT.dId = d_id;

  SELECT str1, str2, city, state, zip, name
  INTO d_str1, d_str2, d_city, d_state, d_zip, d_name
  FROM DISTRICT D
  WHERE D.wId = w_id AND D.dId = d_id;

  IF (by_name) THEN
    SELECT COUNT(cId)
    INTO name_cnt
    FROM CUSTOMER C
    WHERE C.last = c_last AND C.dId = c_d_id AND
    C.wId = c_w_id;
```

```
  DECLARE
  c_byname CURSOR FOR
    SELECT first, middle, cId, str1, str2, city, state, zip,
    phone, credit, creditLim, discount,balance, since
    FROM CUSTOMER C
    WHERE C.wId = c_w_id AND C.dId = c_d_id AND
    C.last = c_last
    ORDER BY C.first;

  OPEN c_byname;

  -- Locate midpoint customer
  IF (name_cnt%2 = 1) THEN
    name_cnt := name_cnt + 1; --
  END IF;
  FOR n in 0 ..namecnt/2 LOOP
    FETCH c_byname
    INTO c_first,c_middle,c_id, c_str1, c_str2, c_city, c_state,c_zip,
       c_phone, c_credit, c_creditLim, c_discount,c_balance,c_since
  END LOOP;

  CLOSE c_byname;

ELSE
  SELECT first, middle, last, str1, str2, city, state, zip, phone,
  credit, creditLim, discount,balance, since
  INTO c_first,c_middle,c_last, c_str1, c_str2, c_city, c_state, c_zip,
      c_phone, c_credit, c_creditLim, c_discount,c_balance,c_since
  FROM CUSTOMER C
  WHERE C.wId = c_w_id AND C.dId = c_d_id AND
  C.cId = c_id;
END IF;

c_balance := c_balance - h_amount;
```

```
c_credit[2] := '\0'; -- insert null char at the end

UPDATE CUSTOMER SET YTDPaymt = YTDPaymt + h_amount
WHERE CUSTOMER.wId = c_w_id AND CUSTOMER.dId = c_d_id AND CUSTOMER.cId
   = c_id;

UPDATE CUSTOMER SET paymtCnt = paymtCnt + 1
WHERE CUSTOMER.wId = c_w_id AND CUSTOMER.dId = c_d_id AND CUSTOMER.cId
   = c_id;

IF (c_credit LIKE '%BC%') THEN
  SELECT data into c_data
  FROM CUSTOMER C
  WHERE C.wId = c_w_id AND C.dId = c_d_id AND C.cId = c_id;

  EXEC FORMAT(c_new_data,"|␣%s␣%s␣%s␣%s␣%s␣$%s␣%s␣%s",
  c_id,c_d_id,c_w_id,d_id,w_id,h_amount,h_date, h_data);
  c_new_data := CONCAT(c_new_data,c_data);

  UPDATE CUSTOMER SET balance = c_balance, data = c_new_data
  WHERE CUSTOMER.wId = c_w_id AND CUSTOMER.dId = c_d_id AND
  CUSTOMER.cId = c_id;
ELSE
  UPDATE CUSTOMER SET balance = c_balance
  WHERE CUSTOMER.wId = c_w_id AND CUSTOMER.dId = c_d_id AND
  CUSTOMER.cId = c_id;
END IF;

h_data := w_name
h_data[10] := '\0'
h_data = CONCAT(h_data,d_name)
-- delete end of string after shifting
h_data[20]='␣';
h_data[21]='␣';
h_data[22]='␣';
```

```
  h_data[23]='␣';

  INSERT INTO HISTORY(cDId, cWId, cId, dId, wId, date, amount, data)
  VALUES (c_d_id, c_w_id, c_id, c_d_id, c_w_id, datetime, c_amount,
     c_data);

  RETURN 0;
END;

$$ LANGUAGE PLpgSQL;
```

4.2.3 The Transaction Profile

For a given warehouse number w_id, district number d_id, customer number c_id or customer last name c_last, payment amount h_amount, and indicator by_name:

1. The row in the WAREHOUSE (W) table with matching W.wId is selected. W.wYTD (the warehouse's year-to-date balance) is increased by h_amount. W.name, W.str1, W.str2, W.city, and W.zip are retrieved.

2. The row in the DISTRICT (D) table with matching D.wId and D.dId is selected. D.name, D.str1, D.str2, D.city, D.state, and D.zip are retrieved and D.dYtd (the district's year-to-date balance) is increased by h_amount.

3. There are two cases possible (indicated by the boolean by_name):

   (a) **Case 1:** the customer is selected based on customer last name.
   All rows in the CUSTOMER (C) table with matching C.wId, C.dId and C.last are selected sorted by C.first in ascending order. Let n be the number of rows selected. C.cId, C.first, C.middle, C.str1, C.str2, C.city, C.state, C.zip, C. phone, C.since, C.credit, C.credLim, C.discount, and C.balance are retrieved from the row at position (n/2 rounded up to the next integer) in the sorted set of selected rows from the CUSTOMER table. C.balance is decreased by h_amount.

   (b) **Case 2:** the customer is selected based on customer number.
   The row in the CUSTOMER table with matching C.wId, C.dId, and C.cId is selected. C.first, C.middle, C.last, C.str1, C.str2, C.city, C.state, C.zip,

C.phone, C.since, C.credit, C.creditLim, C.discount, and C.balance are retrieved. C.balance is decreased by h_amount.

4. In both cases, the C.YTDPaymt is increased by h_amount, and the C.PaymtCnt is incremented by 1.

5. If the value of C.credit is equal to "BC", then C.data is also retrieved from the selected customer and the following history information: C.cId, C.dId, C.wId, D.dId, W.wId, and h_amount, are inserted at the left of the C.data field by shifting the existing content of C.data to the right by an equal number of bytes and by discarding the bytes that are shifted out of the right side of the C.data field. The content of the C.data field never exceeds 500 characters. The selected customer is updated with the new C.data field. If C.data is implemented as two fields, they must be treated and operated on as one single field.

6. The function FORMAT used in the code is similar to the C function sprintf or the Python format() function. It substitutes the placeholders in the string (the second argument) with string values of the variables that follow it.

4.3 The Order-Status Transaction

4.3.1 Overview

The Order-Status business transaction queries the status of a customer's last order.

4.3.2 The osstat Function

```
CREATE OR REPLACE FUNCTION osstat(by_name BOOLEAN, c_last VARCHAR(16),
   d_Id INTEGER, w_Id INTEGER, c_id INTEGER)

RETURNS INTEGER AS $$

DECLARE
  name_cnt INTEGER;
  o_id INTEGER;
  o_carrier_id INTEGER;
  ent_date DATE;
  c_balance NUMERIC(12,2);
```

```
  c_first VARCHAR(16);
  c_middle CHAR(2);
  c_last VARCHAR(16);
  i INTEGER := 0;

BEGIN
  IF (by_name) THEN
    SELECT COUNT(cId) INTO name_cnt
    FROM CUSTOMER C
    WHERE C.last = c_last AND C.dId = d_Id AND C.wId = w_Id;

    DECLARE
    c_name CURSOR FOR
      SELECT balance, first, middle, cId
      FROM CUSTOMER C
      WHERE C.last = c_last AND C.dId = d_id AND C.wId = w_id
      ORDER BY C.first;

    OPEN c_name;

    -- locate midpoint customer
    IF (name_cnt%2 = 1) THEN
      name_cnt := name_cnt + 1;
    END IF;
    FOR n in 0 ..name_cnt/2 LOOP
      FETCH c_name
      INTO c_balance, c_first, c_middle, c_id
    END LOOP;

    CLOSE c_name;

  ELSE
    SELECT balance, first, middle, last
    INTO c_balance, c_first, c_middle, c_last
    FROM CUSTOMER C
```

```
    WHERE C.cId = c_id AND C.dId = d_id AND C.wId = w_Id;
  END IF;

  SELECT oId, carrierId, entryDate
  INTO o_id, o_carrier_id, ent_date
  FROM ORDERS O
  ORDER BY O.oId DESC;

  DECLARE
  c_line CURSOR FOR
    SELECT iId, supplyWId, quantity, amount, deliveryDate
    FROM ORDERLINE OL
    WHERE OL.oId = o_id AND OL.dId = d_id AND OL.wId = w_id;

  OPEN c_line;

  WHILE (sql_notfound = FALSE)
    i := i + 1;
    FETCH c_line
    INTO ol_i_id[i], ol_supply_w_id[i], ol_quantity[i], ol_amount[i],
       ol_delivery_d[i]
  END WHILE;

  CLOSE c_line;
  RETURN 0;
END;
$$ LANGUAGE PLpgSQL;
```

4.3.3 The Transaction Profile

For a given customer number c_id and associated district number d_id, warehouse number w_id:

1. **Case 1**, the customer is selected based on customer last name: all rows in the CUSTOMER (C) table with matching C.wId, C.dId, and C.last are selected sorted by C.first in ascending order. Let n be the number of rows selected. C.balance,

C.first, C.middle, and C.last are retrieved from the row at position n/2 rounded up in the sorted set of selected rows from the CUSTOMER table.

2. **Case 2**, the customer is selected based on customer number: the row in the CUSTOMER table with matching C.wId, C.dId, and C.last is selected and C.balance, C.first, C.middle, and C.last are retrieved.

3. The row in the ORDER (O) table with matching O.wId (equals C.wId), O.dId (equals C.dId), O.cId (equals C.cId), and with the largest existing O.oId, is selected. This is the most recent order placed by that customer. O.oId, O.carrierId, O.entryDate are retrieved.

4. All rows in the ORDERLINE (OL) table with matching OL.wId (equals O.wId), OL.dId (equals O.dId), and OL.oId (equals O.oId) are selected and the correspnding sets of OL.iId, OL.supplyWId, OL.quantity, OL.amount, and OL.deliveryDate are retrieved.

4.4 The Delivery Transaction

4.4.1 Overview

The delivery business transaction consists of processing a batch of 10 new (not yet delivered) orders. Each order is processed (delivered) in full within the scope of a read-write database transaction. The Delivery transaction must be executed in deferred mode (more on page 40 of TPC-C document).

4.4.2 The delivery Function

```
CREATE OR REPLACE FUNCTION delivery (w_id INTEGER,d_id INTEGER,
   o_carrier_id INTEGER)
RETURNS INTEGER AS $$

DECLARE
  no_o_id INTEGER;
  c_id INTEGER;
  ol_total NUMERIC(6,2);
  date_time TIMESTAMP := current_timestamp;
```

```
BEGIN

  FOR d_id in 1 ..DIST_PER_WARE LOOP -- constant DIST_PER_WARE = 10

    DECLARE
    c_no CURSOR FOR
      SELECT oId
      FROM NEW_ORDER NO
      WHERE NO.dId = d_id AND NO.wId = w_id
      ORDER BY NO.oId ASC;

    OPEN c_no;

    FETCH c_no INTO no_o_id;

    DELETE FROM NEW_ORDER WHERE CURRENT OF c_no;
    CLOSE c_no;

    SELECT cId INTO c_id FROM ORDERS O
    WHERE O.oId = no_o_id AND O.dId = d_id AND O.wId = w_id;

    UPDATE ORDERS SET carrierId = o_carrier_id
    WHERE ORDERS.oId = no_o_id AND ORDERS.dId = d_id AND ORDERS.wId =
       w_id;

    UPDATE ORDERLINE SET deliveryDate = date_time
    WHERE ORDERLINE.oId = no_o_id AND ORDERLINE.dId = d_id AND ORDERLINE.
       wId = w_id;

    SELECT SUM(amount) INTO ol_total
    FROM ORDERLINE OL
    WHERE OL.oId = no_o_id AND OL.dId = d_id AND OL.wId = w_id;

    UPDATE CUSTOMER SET balance = balance + ol_total
```

```
    WHERE CUSTOMER.cId = c_id AND CUSTOMER.dId = d_id AND CUSTOMER.wId =
        w_Id;

    -- added per transaction profile
    UPDATE CUSTOMER SET deliveryCnt = deliveryCnt + 1
    WHERE CUSTOMER.cId = c_id AND CUSTOMER.dId = d_id AND CUSTOMER.wId =
        w_Id;

  END LOOP;
  RETURN 0;
END;

$$ LANGUAGE PLpgSQL;
```

4.4.3 The Transaction Profile

For a given warehouse number w_id, for each of the 10 districts d_id within that warehouse, and for a given carrier number o_carrier_id:

1. The row in the NEW-ORDER (NO) table with matching NO.wId (equals w_id) and NO.dId (equals d_id) and with the lowest NO.oId value is selected. This is the oldest undelivered order of that district. NO.oId (the order number) is retrieved. If no matching row is found, then the delivery of an order for this district is skipped (more on page 42 of the document).

2. The selected row in the NEW-ORDER table is deleted.

3. The row in the ORDER (O) table with matching O.wId (equals w_id), O.dId (equals d_id), and O.oId (equals no_o_id) is selected, O.cId (the customer number) is retrieved, and O.carrierId is updated.

4. All rows in the ORDER-LINE (OL) table with matching OL.wId (equals O.wId), OL.dId (equals O.dId),and OL.oId (equals O.oId) is selected. All OL.deliveryDate (the delivery dates) are updated to the current system time as returned by the operating system and the sum of all OL.amount is retrieved.

5. The row in the CUSTOMER (C) table with matching C.wId (equals w_id), C.dId (equals d_id),and C.cId (equals O.cId) is selected and C.balance is increased by the

sum of all order-line amounts (OL.amount) previously retrieved. C.deliveryCnt is incremented by 1.

4.5 The Stock-Level Transaction

4.5.1 Overview

The Stock-Level transaction determines the number of recently sold items that have a stock level below a specified threshold.

4.5.2 The stocklevel Function

```
tCREATE OR REPLACE FUNCTION stocklevel(w_id INTEGER,d_id INTEGER,
   threshold) {
RETURNS INTEGER AS $$

DECLARE
  o_id INTEGER;
  stock_count INTEGER;
BEGIN
  SELECT nextOrdId INTO o_id
  FROM DISTRICT D
  WHERE D.wId = w_id AND D.dId = d_id;

  SELECT COUNT(DISTINCT(S.iId)) INTO stock_count
  FROM ORDERLINE OL, STOCK S
  WHERE OL.wId = w_id AND OL.dId = d_id AND OL.oId < o_id AND OL.oId >=
     o_id - 20
  AND S.wId = w_id AND S.iId = OL.iId AND S.quantity < threshold;

  RETURN 0;
END;
$$ LANGUAGE PLpgSQL;
```

4.5.3 The Transaction Profile

For a given warehouse number (w_id), district number (d_id), and stock level threshold (threshold):

1. The row in the DISTRICT (D) table with matching D.wId and D.dId is selected and D.nextOrdId is retrieved.
2. All rows in the ORDER-LINE (OL) table with matching OL.wId (equals w_id), OL.dId (equals d_id), and OL.oId (lower than D.nextOrdId and greater than or equal to D.nextOrdId - 20) are selected. They are the items for 20 recent orders of the district.
3. All rows in the STOCK (S) table with matching S.iId (equals OL.iId) and S.wId (equals w_id) from the list of distinct item numbers and with S.quantity lower than threshold are counted (giving low_stock).

4.6 Sample Load Program

4.6.1 Overview

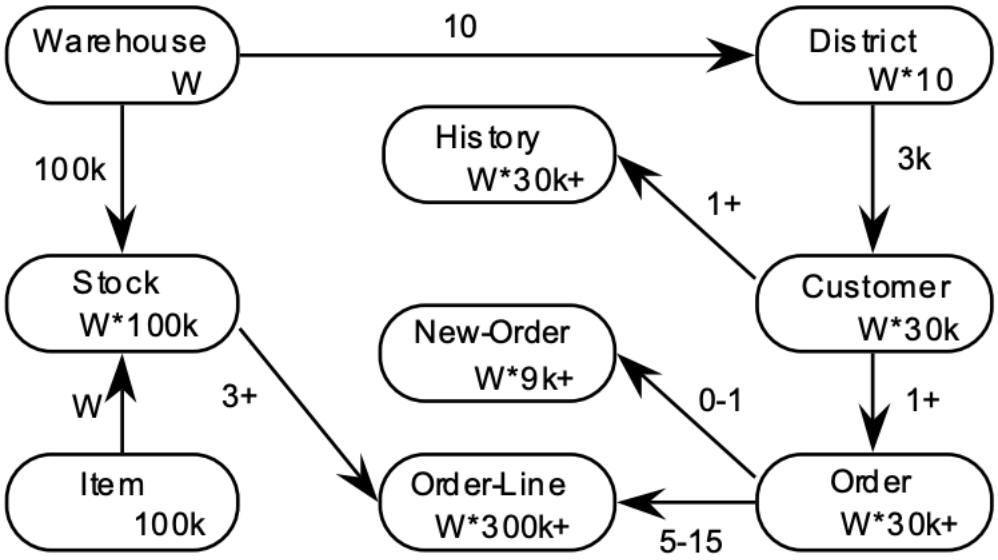
The code for a sample load program is given on page 117-129 of the TPC-C document. It defines the global variables, loads the tables (WAREHOUSE, DISTRICT, CUSTOMER, ORDER-LINE etc.), generates data for them according to the guidelines, and handles errors.

Worth noting are some contants and global variables used in some transactions above:

- The maximum number of items: MAXITEMS = 100000
- Customers per district: CUST_PER_DIST = 3000
- Districts per warehouse: DIST_PER_WARE = 10
- Orders per districts: ORD_PER_DIST = 3000

Appendices

A Database entities & relationships



- The numbers in the entity blocks represent the cardinality of the tables (number of rows). These numbers are factored by W, the number of Warehouses, to illustrate the database scaling.

- The numbers next to the relationship arrows represent the cardinality of the relationships (average number of children per parent).

- The plus (+) symbol is used after the cardinality of a relationship or table to illustrate that this number is subject to small variations in the initial database population over the measurement interval (see Clause 5.5) as rows are added or deleted.

B Table layouts

B.1 WAREHOUSE Table Layout

Field Name	Field Definition	Comments
wId	2*W unique IDs	W Warehouses are populated
name	variable text, size 10	
str1	variable text, size 20	Address line 1
str2	variable text, size 20	Address line 2
city	variable text, size 20	
state	fixed text, size 2	
zip	fixed text, size 9	
tax	signed numeric(4,4)	Sales tax
wYTD	signed numeric(12,2)	Year to date balance

B.2 DISTRICT Table Layout

Field Name	Field Definition	Comments
dId	20 unique IDs	10 are populated per warehouse
wId*	2*W unique IDs	Associated warehouse
name	variable text, size 10	
str1	variable text, size 20	Address line 1
str2	variable text, size 20	Address line 2
city	variable text, size 20	
state	fixed text, size 2	
zip	fixed text, size 9	
tax	signed numeric(4,4)	Sales tax
dYTD	signed numeric(12,2)	Year to date balance
nextOrdId	10,000,000 unique IDs	Next available Order number, increment by 1 every time new order is added

B.3 CUSTOMER Table Layout

Field Name	Field Definition	Comments
cId	96,000 unique IDs	3,000 are populated per district
dId	20 unique IDs	Associated warehouse
wId	2*W unique IDs	Customer-warehouse ID
first	variable text, size 16	First name
middle	fixed text, size 2	Middle initials (max 2)
last	variable text, size 16	Last name
str1	variable text, size 20	Address line 1
str2	variable text, size 20	Address line 2
city	variable text, size 20	
state	fixed text, size 2	
zip	fixed text, size 9	
phone	fixed text, size 16	
since	date and time	
credit	fixed text, size 2	"GC"=good, "BC"=bad
credLim	signed numeric(12, 2)	
discount	signed numeric(4, 4)	
balance	signed numeric(12, 2)	
YTDPaymt	signed numeric(12, 2)	Customer payment year to date
paymtCnt	numeric(4)	Customer payment count
deliveryCnt	numeric(4)	
data	variable text, size 500	Miscellaneous information

B.4 HISTORY Table Layout

Field Name	Field Definition	Comments
cId*	96,000 unique IDs	
cDId*	20 unique IDs	
cWId*	2*W unique IDs	
dId*	20 unique IDs	
wId*	2*W unique IDs	
date	date and time	
amount	signed numeric(6, 2)	
data	variable text, size 24	Miscellaneous information

Comment: Rows in the History table do not have a primary key as, within the context of the benchmark, there is no need to uniquely identify a row within this table.

B.5 NEW-ORDER Table Layout

Field Name	Field Definition	Comments
oId	10,000,000 unique IDs	
dId*	20 unique IDs	
wId*	2*W unique IDs	

B.6 ORDER Table Layout

Field Name	Field Definition	Comments
oId	10,000,000 unique IDs	
dId*	20 unique IDs	
wId*	2*W unique IDs	
cId*	96,000 unique IDs	
entryDate	date and time	
carrierId	10 unique IDs, or null	
oLCnt	numeric(2)	Count of Order-Lines (number of items)
allLocal	numeric(1)	If all local then 1 else 0

B.7 ORDER-LINE Table Layout

Field Name	Field Definition	Comments
oId*	10,000,000 unique IDs	
dId*	20 unique IDs	
wId*	2*W unique IDs	
number	15 unique IDs	
iId	200,000 unique IDs	Item ID
supplyWId	2*W unique IDs	
deliveryDate	date and time, or null	
quantity	numeric(2)	
amount	signed numeric(6, 2)	
distInfo	fixed text, size 24	

B.8 ITEM Table Layout

Field Name	Field Definition	Comments
iId	200,000 unique IDs	100,000 items are populated
iMId	200,000 unique IDs	Image ID associated to Item
name	variable text, size 24	
price	numeric(5, 2)	
data	variable text, size 50	Brand information

B.9 STOCK Table Layout

Field Name	Field Definition	Comments
iId	200,000 unique IDs	100,000 items are populated
wId	2*W unique IDs	
quantity	signed numeric(4)	
dist01	fixed text, size 24	
dist02	fixed text, size 24	
dist03	fixed text, size 24	
dist04	fixed text, size 24	
dist05	fixed text, size 24	
dist06	fixed text, size 24	
dist07	fixed text, size 24	
dist08	fixed text, size 24	
dist09	fixed text, size 24	
dist10	fixed text, size 24	
YTD	numeric(8)	
orderCnt	numeric(4)	
remoteCnt	numeric(4)	
data	variable text, size 50	Information