AuctionMark OLTP Benchmark

Project Report

Visawee Angkanawaraphan

2/18/2011

Table of Contents

AuctionMark OLTP Benchmark

1. Introduction

AuctionMark benchmark is an Online Transaction Processing (OLTP) workload that simulates the activities found in a well-known auction site. This benchmark comprises of 13 tables and 14 stored procedures representing the auction site's core transactions.

There are two types of transactions in AuctionMark: (1) transactions that are initiated by a user and (2) book keeping transactions that are invoked periodically by the system. Sellers and buyers are two main user types in the system. Transaction that are related to sellers include (1) adding an item for auction, (2) updating an existing item's information, and (3) responding to buyers' comments and feedbacks. Buyers related transactions include (1) adding an item to watch list, (2) retrieving item's detail, (3) placing a bid on an open auction, (4) purchasing an item, and (5) leaving comments/feedback for sellers. The system periodically invoke check-winning-bid transaction to check if any auction comes to the end, and checks the winning bid for that item.

This benchmark is implemented in Java, and works with H-Store database. It is designed for scalability and flexibility (a user can configure the proportion of different transactions executed).

The source code is publicly available from the H-Store repository via anonymous SVN:

```
svn co
https://database.cs.brown.edu/svn/hstore/trunk/tests/edu/brown/benchmarks/auctionmark
```

2. Data Tables

The tables are all named in a singular form. Each column is prefixed with an abbreviation of the table that it is a member of. Foreign key references include the abbreviation of the parent column. The only exception to this rule are those tables with more than one column referencing the same foreign key parent table (e.g., ITEM_COMMENT.ic_u_id and ITEM_COMMENT.ic_buyer_id).

2.1 Global Tables

The *Global Tables* represent data that is not specific to one particular item or user.

2.1.1 REGION

Number of Records: 62
Average Tuple Size: 28 bytes
This table represents the geographical regions of users.

Column	Type	Cardinality	References	Description
r_id	BIGINT		-	Region's ID
r_name	VARCHAR	String length 6 – 32	-	Region's name

2.1.2 GLOBAL_ATTRIBUTE_GROUP

Number of Records: 100
Average Tuple Size: 36 bytes
This table represents merchandises' global attribute groups (e.g., brand, material, features).

Column	Type	Cardinality	References	Description
gag_id	BIGINT		-	Global attribute group's ID
gag_c_id	BIGINT		CATEGORY.c_id	Associated Category's ID
gag_name	VARCHAR	String length 6 – 32	-	Global attribute group's name

2.1.3 GLOBAL_ATTRIBUTE_VALUE

Number of Records: # GLOBAL_ATTRIBUTE_GROUP * 10
Average Tuple Size: 36 bytes
This table represents merchandises' global attributes within each attribute groups (e.g., Rolex, Casio, Seiko within brand).

Column	Type	Cardinality	References	Description
gav_id	BIGINT		-	Global attribute group's ID
gav_gag_id	BIGINT		GLOBAL_ATTRIBUTE_GROUP.gag_id	Associated Category's ID
gav_name	VARCHAR	String length 6 – 32	-	Global attribute group's name

2.1.4 CATEGORY

Number of Records: 19,459
Average Tuple Size: 85
This table represents merchandises' categories. Category can be hierarchical aligned using c_parent_id. The records in this table were retrieved from a well-known

Column	Type	Cardinality	References	Description
c_id	BIGINT		-	Category's ID
c_name	VARCHAR		-	Category's name
c_parent_id	BIGINT		CATEGORY.c_id	Parent category's ID

2.2 User Tables

2.2.1 USER

Number of Records: 1,000,000
Average Tuple Size: 376 bytes
This table represents users of the system both sellers and buyers.

Column	Type	Cardinality	References	Description
u_id	BIGINT		-	User's ID
u_rating	BIGINT	Integer 0 – 6	-	User's rating as a seller
u_balance	DOUBLE		-	User's balance
u_created	TIMESTAMP		-	User's create date
u_r_id	BIGINT	62	-	User's region ID
u_sattr0	VARCHAR	String length 16 – 64	-	User's attribute 0
u_sattr1	VARCHAR	String length 16 – 64	-	User's attribute 1
u_sattr2	VARCHAR	String length 16 – 64	-	User's attribute 2
u_sattr3	VARCHAR	String length 16 – 64	-	User's attribute 3
u_sattr4	VARCHAR	String length 16 – 64	-	User's attribute 4
u_sattr5	VARCHAR	String length 16 – 64	-	User's attribute 5
u_sattr6	VARCHAR	String length 16 – 64	-	User's attribute 6
u_sattr7	VARCHAR	String length 16 – 64	-	User's attribute 7

2.2.2 USER_ATTRIBUTES

Number of Records: # USER x 1.3
Average Tuple Size: 62 bytes
This table represents attributes of users.

Column	Type	Cardinality	References	Description
ua_id	BIGINT		-	User attribute's ID
ua_u_id	BIGINT		USER.u_id	User's ID
ua_name	VARCHAR	String length 5 – 32	-	User attribute name
ua_value	VARCHAR	String length 5 – 32	-	User attribute value
u_created	TIMESTAMP		-	Create timestamps

2.3 Item Tables

2.3.1 ITEM

Number of Records: # USER x 10
Average Tuple Size: 395 bytes
The ITEM id consists of a composite key where the lower 48-bits of the number is the USER.u id and the upper 16-bits is the auction count for that user.

Column	Type	Cardinality	References	Description
i_id	BIGINT		-	Item's ID
i_u_id	BIGINT		USER.u_id	Seller's ID
i_c_id	BIGINT		CATEGORY.c_id	Category's ID
i_name	VARCHAR	String length 6 – 32	-	Item's name
i_description	VARCHAR	String length 50 – 255	-	Item's description
i_user_attributes	VARCHAR	String length 20 – 255	-	Text field for attributes defined just for this item

Column	Type	Cardinality	References	Description
i_initial_price	DOUBLE		-	Item’s initial price
i_current_price	DOUBLE		-	Item’s current price
i_num_bids	BIGINT		-	Item’s number of bids
i_num_images	BIGINT		-	Item’s number of images
i_num_global_attrs	BIGINT		-	Item’s number of global attributes
i_start_date	TIMESTAMP		-	Item’s bid start date
i_end_date	TIMESTAMP		-	Item’s bid end date
i_status	INT		-	Item’s status (0 = OPEN, 1 = WAITING_FOR_PURCHASE, 2 = CLOSED)

2.3.2 ITEM_IMAGE

Number of Records: # ITEM x 3.21
Average Tuple Size: 85 bytes
This table keeps paths to the images of items in the ITEM table.

Column	Type	Cardinality	References	Description
ii_id	BIGINT		-	Image’s ID
ii_i_id	BIGINT		ITEM.i_id	Item’s ID
ii_u_id	BIGINT		ITEM.i_u_id	Item’s user ID
ii_path	VARCHAR	String length 20 – 100	-	Image’s path

2.3.3 ITEM_COMMENT

Number of Records: # ITEM x 0.466
Average Tuple Size: 180 bytes
Comments of items in the ITEM table.

Column	Type	Cardinality	References	Description
ic_id	BIGINT		-	Comment's ID
ic_i_id	BIGINT		ITEM.i_id	Item's ID
ic_u_id	BIGINT		ITEM.i_u_id	Item's user ID
ic_buyer_id	BIGINT		USER.u_id	Item's buyer ID
ic_date	TIMESTAMP		-	Comment's create date
ic_question	VARCHAR	String length 10 – 128	-	Comment from buyer
ic_response	VARCHAR	String length 10 – 128	-	Response from seller

2.3.4 ITEM_FEEDBACK

Number of Records: # ITEM x 0.466
Average Tuple Size: 180 bytes
Feedbacks of items in the ITEM table.

Column	Type	Cardinality	References	Description
if_id	BIGINT		-	Feedback's ID
if_i_id	BIGINT		ITEM.i_id	Item's ID
if_u_id	BIGINT		ITEM.i_u_id	Item's user ID
if_buyer_id	BIGINT		USER.u_id	Item's buyer ID
if_rating	BIGINT		-	Rating given to the seller
if_date	TIMESTAMP		-	Feedback date

Column	Type	Cardinality	References	Description
if_comment	VARCHAR	String length 10 – 128	-	Feedback from buyer

2.3.5 ITEM_BID

Number of Records: # ITEM x 10.62
Average Tuple Size: 64 bytes
Bids of items in the ITEM table.

Column	Type	Cardinality	References	Description
ib_id	BIGINT		-	Bid’s ID
ib_i_id	BIGINT		ITEM.i_id	Item’s ID
ib_u_id	BIGINT		ITEM.i_u_id	Item’s seller ID
ib_buyer_id	BIGINT		USER.u_id	Buyer’s ID
ib_bid	DOUBLE		-	Bid
ib_max_bid	DOUBLE		-	Max bid
ib_created	TIMESTAMP		-	Bid’s create date
ib_updated	TIMESTAMP		-	Bid’s update date

2.3.6 ITEM_MAX_BID

Number of Records: # ITEM x 0.76
Average Tuple Size: 56 bytes
The maximum bids of each item in the ITEM table.

Column	Type	Cardinality	References	Description
imb_i_id	BIGINT		ITEM.i_id	Item’s ID
imb_u_id	BIGINT		ITEM.i_u_id	Item’s seller ID
imb_ib_id	BIGINT		ITEM_BID.ib_id	Bid’s ID
imb_ib_i_id	BIGINT		ITEM_BID.ib_i_id	Bid’s item ID
imb_ib_u_id	BIGINT		ITEM_BID.ib_u_id	Bid’s seller ID
imb_created	TIMESTAMP		-	Max bid create timestamps
imb_updated	TIMESTAMP		-	Max bid update timestamps

2.3.7 ITEM_PURCHASE

Number of Records: # ITEM x 0.026
Average Tuple Size: 40 bytes
Tthe purchasing records from the buyers.

Column	Type	Cardinality	References	Description
ip_id	BIGINT		-	Purchase’s ID
ip_ib_id	BIGINT		ITEM_BID.ib_id	Bid’s ID
ip_i_id	BIGINT		ITEM_BID.ib_i_id	Item’s ID
ip_u_id	BIGINT		ITEM_BID.ib_u_id	Item’s seller ID
ip_date	TIMESTAMP		-	Purchase timestamps

2.3.8 USER_ITEMS

Number of Records: # ITEM x 0.026
Average Tuple Size: 32 bytes
The items that a user has recently purchased.

Column	Type	Cardinality	References	Description
ui_u_id	BIGINT		USER.u_id	User’s ID
ui_i_id	BIGINT		ITEM.i_id	Item’s ID
ui_seller_id	BIGINT		ITEM.i_u_id	Seller’s ID
ui_created	TIMESTAMP		-	Purchase timestamps

3. Stored Procedures

The stored procedures below are executed in the following distribution except for two stored procedures; check-winning-bid and post-auction. check-winning-bid is executed every 10 seconds, and post-auction is executed right after check-winning-bid.

Stored procedures	Frequency
new-user	5%
new-item	10%
new-bid	18%
new-comment	2%
new-comment-response	1%
new-purchase	2%
new-feedback	3%
get-item	40%
update-item	2%
get-comment	2%
get-user-info	10%
get-watched-item	5%

3.1 new-user

Creates a new USER record. The rating and balance are both set to zero.

The benchmark randomly selects id from a pool of region ids as an input for u_r_id parameter using flat distribution.

Input parameters

Name	Type	Description
u_id	BIGINT	The new user's unique id
u_r_id	BIGINT	User's region
attributes[]	8 × VARCHAR(12-64)	An array of random string attributes for this user

Output parameters

None

Pseudo code

```
// Insert the user
INSERT INTO USER (u_id, u_rating, u_balance, u_created, u_r_id, u_sattr0, u_sattr1, u_sattr2, u_sattr3, u_sattr4, u_sttar5, u_sattr6, u_sattr 7)
VALUES (<u_id>, 0, 0, <current_timestamp>, <u_r_id>, <attributes[0]>, <attributes[1]>, <attributes[2]>, <attributes[3]>, <attributes[4]>, <attributes[5]>, <attributes[6]>, <attributes[7]>);
```

3.2 new-item

Insert a new ITEM record for a user. The benchmark client provides all of the preliminary information required for the new item, as well as optional information to create derivative image and attribute records. After inserting the new ITEM record, the transaction then inserts any GLOBAL ATTRIBUTE VALUE and ITEM IMAGE. The unique identifier for each of these records is a composite 64-bit key where the lower 60-bits are the i id parameter and the upper 4-bits are used to represent the index of the image/attribute. For example, if the i id is 100 and there are four items, then the composite key will be 0 100 for the first image, 1 100 for the second, and so on. After these records are inserted, the transaction then updates the USER record to add the listing fee to the seller's balance.

The benchmark randomly selects id from a pool of users as an input for u_id parameter using Gaussian distribution. A c_id parameter is randomly selected using a flat histogram from the real auction site's item category statistic.

Input parameters

Name	Type	Description
i_id	BIGINT	The new item's unique id (0–2^60)
u_id	BIGINT	The seller's user id
c_id	BIGINT	The new item's category id
name	VARCHAR	The title of the new item
description	VARCHAR	The text description of the new item
initial_price	DOUBLE	The initial price of the new item
reserve price	DOUBLE	The seller's reserve selling price (can be null)
buy now	DOUBLE	The seller's "instance purchase" price (can be null)
attributes	VARCHAR	Textual BLOB of user-defined attributes for the item
gag_ids[]	(0-16) × BIGINT	List of GLOBAL ATTRIBUTE GROUP ids to include for this item
gav_ids[]	(0-16) × BIGINT	List of GLOBAL ATTRIBUTE VALUE ids to include for this item
images[]	(0-16) × VARCHAR(128)	List of unique image paths
start_date	TIMESTAMP	Auction start timestamp
end_date	TIMESTAMP	Auction end timestamp

Output parameters

Name	Type	Description
i_id	BIGINT	The new item's unique id
i_u_id	BIGINT	The seller's user id

Pseudo code

```
// Get attribute names and append them to the item description
description = “”;
for i = 0 to <gag_ids.lenght> – 1
    description += SELECT gag_name + ” ” + gav_name FROM GLOBAL_ATTRIBUTE_GROUP, GLOBAL_ATTRIBUTE_VALUE
                   WHERE gav_id = gav_ids[i] AND gav_gag_id = gag_ids[i] AND gav_gag_id = gag_id;

// Insert the item
INSERT INTO ITEM (i_id, i_u_id, i_c_id, i_name, i_description, i_user_attributes, i_initial_price, i_num_bids, i_num_images,
i_num_global_attrs, i_start_date, i_end_date)
VALUES (<i_id>, <u_id>, <c_id>, <name>, <description>, <attributes>, <initial_price>, 0, <images.length>, <gav_ids.length>,
<start_date>, <end_date>);

// Add item images
for i = 0 to <images.length> – 1
    ii_id = (i << 60) | (<i_id> & 0x0FFFFFFFFFFFFFFF);
    INSERT INTO ITEM_IMAGE (ii_id, ii_i_id, ii_u_id, ii_path)
    VALUES (<ii_id>, <i_id>, <u_id>, <images[i]>);

// Decrement user balance
UPDATE USER
SET u_balance = u_balance – 1
WHERE u_id = <u_id>;
```

3.3 new-bid

A buyer enters in a new bidding for an open item. If there is no existing bid on the item, the new bid is marked as the winning bid and the item is updated accordingly. If there is an existing highest bid, then the transaction has to check whether the new bid amount is greater than the existing bid. The id for the new ITEM_BID record is the ITEM.i_num_bids + 1.
Things to check:
• The auction is still open.
• The potential buyer is not the seller.
• The new bid is greater than the current bid.

The benchmark randomly selects ids from a pool of users as an input for u_id and i_buyer_id parameter using Gaussian distribution.

Input parameters

Name	Type	Description
i_id	BIGINT	The item that is being bidded on

Name	Type	Description
u_id	BIGINT	The seller's user id
i_buyer_id	BIGINT	The buyer's user id
bid	DOUBLE	The amount the buyer bids on the item
maxBid	DOUBLE	The maximum amount the buyer bids on the item

Output parameters

Name	Type	Description
ib_id	BIGINT	The new bid's id

Pseudo code

```
// Increment the number of bids
UPDATE ITEM
SET i_num_bids = i_num_bids + 1
WHERE i_id = <i_id> AND i_u_id = <u_id> AND i_status = 0;

ib_id = SELECT MAX(ib_id) + 1 FROM ITEM_BID WHERE ib_i_id = <i_id> AND ib_u_id = <u_id>;

imb_ib_id = SELECT imb_ib_id FROM ITEM_MAX_BID WHERE imb_i_id = <i_id> AND imb_u_id = <u_id>;

if imb_ib_id is not null then
   // If we have an existing max bid, then we need to figure out whether we are
   // the new highest bidder or if the existing one just has their max_bid bumped up

   current_bid, current_max_bid = SELECT ib_bid, ib_max_bid
                                  FROM ITEM_BID
                                  WHERE ib_id = <ib_id> AND ib_i_id = <i_id> AND ib_u_id = <u_id>;

   newBidWin = false;

   if maxBid > current_max_bid then
      newBidWin = true
      if bid < current_max_bid then
         bid = current_max_bid
      endif
   else
      if bid > current_bid then
         UPDATE ITEM_BID
         SET ib_bid = <bid>
```

```
        WHERE ib_id = <imb_ib_id> AND ib_i_id = <i_id> AND ib_u_id = <u_id>;
      endif
    endif

    INSERT INTO ITEM_BID (ib_id, ib_i_id, ib_u_id, ib_buyer_id, ib_bid, ib_max_bid, ib_created, ib_updated)
    VALUES (<ib_id>, <i_id>, <u_id>, <i_buyer_id>, <bid>, <maxBid>, <current_timestamp>, <current_timestamp>);

    if newBidWin == true then
      UPDATE ITEM_MAX_BID
      SET imb_ib_id = <imb_ib_id>,
      imb_ib_i_id = <i_id>,
      imb_ib_u_id = <u_id>,
      imb_updated = <current_timestamp>
      WHERE imb_i_id = <i_id> AND imb_u_id = <u_id>;
    endif

else
    // There is no existing max bid record, therefore we can just insert ourselves

    INSERT INTO ITEM_BID (ib_id, ib_i_id, ib_u_id, ib_buyer_id, ib_bid, ib_max_bid, ib_created, ib_updated)
    VALUES (<ib_id>, <i_id>, <u_id>, <i_buyer_id>, <bid>, <maxBid>, <current_timestamp>, <current_timestamp>);

    INSERT INTO ITEM_MAX_BID (imb_i_id, imb_u_id, imb_ib_id, imb_ib_i_id, imb_ib_u_id, imb_created, imb_updated)
    VALUES (<i_id>, <u_id>, <ib_id>, <i_id>, <u_id>, <current_timestamp>, <current_timestamp>);

endif
```

3.4 new-comment

A potential buyer posts a question/comment about a seller's item. Each transaction inserts a new ITEM COMMENT record.

The benchmark randomly selects item from a pool of items that are in CLOSED status using flat distribution. It then extracts i_id, seller_id, and buyer_id from the item and associated winning bid as inputs to this stored procedure.

Input parameters

Name	Type	Description
i_id	BIGINT	The item that the potential buyer is commenting on
seller_id	BIGINT	The seller's user id
buyer_id	BIGINT	The potential buyer's user id
question	VARCHAR(128)	The new question posted for the item

Output parameters

Name	Type	Description
ic_id	BIGINT	The new comment's id

Pseudo code

```
INSERT INTO ITEM_COMMENT (ic_id, ic_i_id, ic_u_id, ic_buyer_id, ic_date, ic_question)
VALUES (
(SELECT MAX(ic_id) FROM ITEM_COMMNET WHERE ic_i_id = <i_id> AND ic_u_id = <seller_id>) + 1,
i_id, buyer_id, <current_timestamp>, question);
```

3.5 new-comment-response

The seller responds to an open question/comment created by new-comment-response.

The benchmark randomly selects comment from a pool of comments by users using flat distribution. It then uses i_id, i_c_id, and seller_id from the selected comment as inputs to this stored procedure.

Input parameters

Name	Type	Description
i_id	BIGINT	The item
i_c_id	BIGINT	The item comment record that the seller is replying to
seller_id	BIGINT	The seller's user id
response	VARCHAR(128)	The response to the existing comment/question

Output parameters

None

Pseudo code

```
UPDATE ITEM_COMMENT
SET ic_response = <response>
WHERE ic_id = <i_c_id> AND ic_i_id = <i_id> AND ic_u_id = <seller_id>
```

3.6 new-purchase

The buyer purchases the item for a previously won auction. This stored procedure checks first if the input bid is the winning bid. It then inserts a record to the purchase table, and changes item's status to CLOSED.

The benchmark randomly selects item from a pool of items that are in WAITING_FOR_PURCHASE status using flat distribution. It then gets a wining bid of that item, and extracts ib_id, i_id, u_id, and buyer_id as inputs to this stored procedure.

Input parameters

Name	Type	Description
ib_id	BIGINT	The winning bid ID
i_id	BIGINT	The ID of the purchasing item
u_id	BIGINT	The seller's user ID
buyer_id	BIGINT	The buyer's user ID

Output parameters

Name	Type	Description
ip_id	BIGINT	The new purchase id
ip_ib_id	BIGINT	The associated bid's id
ip_ib_i_id	BIGINT	The associated item's id
u_id	BIGINT	The associated seller's user id
ip_ib_u_id	BIGINT	The associated buyer's user id

Pseudo code

```
imb_ib_id, imb_ib_i_id, imb_ib_u_id = SELECT imb_ib_id, imb_ib_i_id, imb_ib_u_id
                                      FROM ITEM_MAX_BID
                                      WHERE imb_i_id = <i_id> AND imb_u_id = <u_id>;

ib_buyer_id = SELECT ib_buyer_id
              FROM ITEM_BID
              WHERE ib_id = <imb_ib_id> AND ib_i_id = <imb_ib_i_id> AND ib_u_id = <imb_ib_u_id>;

// Validate inputs
if ib_id != imb_ib_id || buyer_id != ib_buyer_id then
   throws error
endif

ip_id = (SELECT MAX(ip_id) FROM ITEM_PURCHASE WHERE ip_ib_id = <ib_id> AND ip_ib_i_id = <i_id> AND ip_ib_u_id =
<u_id>) + 1;

// Insert a new purchase
INSERT INTO ITEM_PURCHASE (ip_id, ip_ib_id, ip_ib_i_id, ip_ib_u_id, ip_date)
VALUES(<ip_id>, <ib_id>, <i_id>, <u_id>, <current_timestamp>);

// Update item status to closed (i_status = 2)
UPDATE ITEM
SET i_status = 2
WHERE i_id = <i_id> AND i_u_id = <u_id>;
```

3.7 new-feedback

The buyer adds a seller feedback for a previously won auction. The rating is either -1, 0, or 1 and is added to the USER.u rating attribute.

The benchmark randomly selects item from a pool of items that are in CLOSED status using flat distribution. It then extracts i_id, seller_id, and buyer_id from the item and associated winning bid as inputs to this stored procedure.

Input parameters

Name	Type	Description
i_id	BIGINT	The purchased item
seller_id	BIGINT	The seller's user id
buyer id	BIGINT	The potential buyer's user id
rating	BIGINT	[−1, 0, 1]

Name	Type	Description
comment	VARCHAR(128)	An optional comment for the seller

Output parameters

Name	Type	Description
if_id	BIGINT	The new feedback's id

Pseudo code

```
if_id = (SELECT MAX(if_id) FROM ITEM_FEEDBACK WHERE if_i_id = <i_id> AND if_u_id = <seller_id>) + 1;

INSERT INTO ITEM_FEEDBACK (if_id, if_i_id, if_u_id, if_buyer_id, if_rating, if_date, if_comment)
VALUES(<if_id>, <i_id>, <seller_id>, <buyer_id>, <rating>, <current_timestamp>, <comment>);
```

3.8 get-item

Get item information. Returns all of the attributes for a particular item.

The benchmark randomly selects item from a pool of items that are in OPEN status using flat distribution. It then extracts i_id and i_u_id from the item as inputs to this stored procedure.

Input parameters

Name	Type	Description
i_id	BIGINT	The id of the item to retrieve
i_u_id	BIGINT	The seller's user id for this item

Output parameters

Name	Type	Description
i_id	BIGINT	The new item's unique id

Name	Type	Description
i_u_id	BIGINT	The seller's user id
i_initial_price	DOUBLE	The initial price of this item
i_current_price	DOUBLE	Current price of this item

Pseudo code

```
// Get the item
SELECT i_id, i_u_id, i_initial_price, i_current_price
FROM ITEM
WHERE i_id = AND i_u_id = AND i_status = 0;
```

3.9 update-item

The buyer modifies an existing auction that is still available. The transaction will just update the description of the auction. A small percentage of the transactions will be for auctions that are uneditable; when this occurs, the transaction will abort.

The benchmark randomly selects item from a pool of items that are in OPEN status using flat distribution. It then extracts i_id and i_u_id from the item as inputs to this stored procedure.

Input parameters

Name	Type	Description
i_id	BIGINT	The id of the item to modify
i_u_id	BIGINT	The seller's user id for this item
Description	VARCHAR(255)	The new description for the item

Output parameters

None

Pseudo code

```
UPDATE ITEM
SET i_description = <description>
WHERE i_id = <i_id> AND i_u_id = <i_u_id>
```

3.10 check-winning-bids

This stored procedure is called once a minute to examine whether any auctions ended within the last 60 seconds. It returns a list of ITEM.i ids of the auctions that need to be processed. This is a broadcast transaction that must be executed on all nodes.

Input parameters

Name	Type	Description
startTime	TIMESTAMP	Start time of the duration that we want to check for ended auctions
endTime	TIMESTAMP	End time of the duration that we want to check for ended auctions

Output parameters

An array of due item objects each containing the following fields.

Name	Type	Description
i_id	BIGINT	Item’s id
i_u_id	BIGINT	Seller’s id
i_status	BIGINT	Item’s status
imb_ib_id	BIGINT	The winning bid’s id (Can be null if there is no winning bid on this item)
ib_buyer_id	BIGINT	The buyer’s id (Can be null if there is no winning bid on this item)

Pseudo code

```
// Get all due items that are still in status OPEN
dueItems = SELECT i_id, i_u_id, i_status
           FROM ITEM
           WHERE i_start_date BETWEEN AND AND i_status = 0
           LIMIT 100;

// For each item, check if there is a winning bid
for each dueItem in dueItems
   itemId = dueItem.i_id
   userId = dueItem.i_u_id
   itemStatus = dueItem.i_status

   maxBidId = SELECT imb_ib_id
              FROM ITEM_MAX_BID
              WHERE imb_i_id = <itemId> AND imb_u_id = <userId>;

   if maxBidId is not null then

      // Found wining bid, retrieve buyer of the winning bid
      buyerId = SELECT ib_buyer_id
                FROM ITEM_BID
                WHERE ib_id = <maxBidId> AND ib_i_id = <itemId> AND ib_u_id = <userId>;

      userItem = {itemId, userId, itemStatus, maxBidId, buyerId}

   else

      userItem = {itemId, userId, itemStatus, null, null}

   endif

   result.add(userItem);

endfor

return result
```

3.11 post-auction

Each transaction performs post-processing of a given closed auction items. This stored procedure goes through each item and update the item status. If there is no bid for an item, this stored procedure changes the item's status to closed. Otherwise, the status is changed to waiting-for-purchase, and a new record is inserted to the USER_ITEM table. All IDs of the same item across fields must be on the same array index. This transaction is invoked immediately after check-winning-bid if check-winning-bid finds due items.

Input parameters

Name	Type	Description
i_ids	Arrays of BIGINT	The items' IDs
seller_ids	Arrays of BIGINT	The sellers' IDs of the items
buyer_ids	Arrays of BIGINT	The buyers' IDs of the items
ib_ids	Arrays of BIGINT	The bids' IDs of the items

Output parameters

Name	Type	Description
closed	BIGINT	The number of CLOSED items processed
waiting	BIGINT	The number of WAITING_FOR_PURCHASE items processed

Pseudo code

```
// Go through each item and update the item status
// We'll also insert a new USER_ITEM record as needed
for i = 0 to i_ids.length – 1
    ib_id = ib_ids[i]
    i_id = i_ids[i]
    seller_id = seller_ids[i]
    buyer_id = buyer_ids[i]
// No bid on this item – set status to close (i_status = 2)
    if ib_id is null then
        UPDATE ITEM
        SET i_status = 2
        WHERE i_id = <i_id> AND i_u_id = <seller_id>;

    // Has bid on this item – set status to wait for purchase (i_status = 1)
    else
```

```
        UPDATE ITEM
        SET i_status = 1
        WHERE i_id = <i_id> AND i_u_id = <seller_id>;

        INSERT INTO USER_ITEM (ui_u_id, ui_i_id, ui_i_u_id, ui_created)
        VALUES(<buyer_id>, <i_id>, <seller_id>, <current_timestamp>);

    endif
endfor
```

3.12 get-comment

Get all item comments of a seller.

The benchmark randomly selects id from a pool of sellers who has items in CLOSED status as a seller_id parameter using flat distribution.

Input parameters

Name	Type	Description
seller_id	BIGINT	The seller's user ID

Output parameters

An array of comments each containing every field in ITEM_COMMENT table.

Pseudo code

```
SELECT * FROM ITEM_COMMENT
WHERE ic_u_id = <seller_id> AND ic_response IS NULL;
```

3.13 get-user-info

Get a user's items as a seller or a buyer. This stored procedure also retrieves items' feedbacks if specify. The get_seller_items, get_buyer_items, and get_feedback must be either 0 or 1. If get_seller_items and get_buyer_items are 1 at the same time, this stored procedure will retrieve all items of a user as a seller.

The benchmark randomly selects id from a pool of sellers who has items in OPEN status as a u_id parameter using flat histogram.

Input parameters

Name	Type	Description
u_id	BIGINT	The ID of the user
get_seller_items	BIGINT	1 if get user items as a seller
get_buyer_items	BIGINT	1 if get user items as a buyer
get_feedback	BIGINT	1 if want to include items' feedback in the response

Output parameters

A user object containing every field in USER table.
An array of item objects each having the following fields.

Name	Type	Description
i_id	BIGINT	Item's id
i_u_id	BIGINT	Seller's id
i_name	VARCHAR	Item's name
i_current_price	DOUBLE	Item's current price
i_end_date	TIMESTAMP	Item's auction end date
i_status	INT	Item's status

An array of item feedback objects each having the following fields.

Name	Type	Description
if_rating	BIGINT	Rating of the item
if_comment	VARCHAR	Feedback of the item
if_date	TIMESTAMP	Date of the feedback
i_id	BIGINT	Item’s id
i_u_id	BIGINT	Seller’s id
i_name	VARCHAR	Item’s name
i_end_date	TIMESTAMP	Item’s auction end date
i_status	INT	Item’s status
u_id	BIGINT	Seller’s id
u_rating	BIGINT	Seller’s rating
u_sattr0	VARCHAR	Seller’s user attribute 0
u_sattr1	VARCHAR	Seller’s user attribute 1

Pseudo code

```
user_results = SELECT u_id, u_rating, u_balance, u_created, u_sattr0, u_sattr1, u_sattr2, u_sattr3, u_sattr4, r_name
               FROM USER, REGION
               WHERE u_id = <u_id> AND u_r_id = r_id;

if get_seller_items == 1 || get_buyer_items == 1 then
   if get_seller_items == 1 then
      item_results = SELECT i_id, i_u_id, i_name, i_current_price, i_end_date, i_status
                     FROM ITEM
                     WHERE i_u_id = <u_id>
                     ORDER BY i_end_date ASC LIMIT 20;
   else if get_buyer_items == 1 then
      item_results = SELECT i_id, i_u_id, i_name, i_current_price, i_end_date, i_status
                     FROM USER_ITEM, ITEM
                     WHERE ui_u_id = <u_id> AND ui_i_id = i_id AND ui_i_u_id = i_u_id
                     ORDER BY i_end_date ASC LIMIT 10;
   endif
```

```
    if get_feedback == 1 then
        feedback_results = SELECT if_rating, if_comment, if_date, i_id, i_u_id, i_name, i_end_date, i_status, u_id, u_rating, u_sattr0,
u_sattr1
                           FROM ITEM_FEEDBACK, ITEM, USER
                           WHERE if_buyer_id = <u_id> AND if_i_id = i_id AND if_u_id = i_u_id AND if_u_id = u_id " +
                           ORDER BY if_date DESC LIMIT 10;
    endif
endif

return user_results, item_results, feedback_results
```

3.14 get-watched-items

Gets all watched items of a buyer.

The benchmark randomly selects id from a pool of users as a u_id parameter using Gaussian distribution.

Input parameters

Name	Type	Description
u_id	BIGINT	The buyer's user ID

Output parameters

An array of user watched items each having the following fields.

Name	Type	Description
uw_u_id	BIGINT	User's id
i_id	BIGINT	Item's id
i_u_id	BIGINT	Seller's id
i_name	VARCHAR	Item's name
i_current_price	DOUBLE	Item's current price
i_end_date	TIMESTAMP	Item's auction end date
i_status	INT	Item's status

Name	Type	Description
uw_created	TIMESTAMP	Watched item create date

Pseudo code

```
SELECT uw_u_id, i_id, i_u_id, i_name, i_current_price, i_end_date, i_status, uw_created
FROM USER_WATCH, ITEM
WHERE uw_u_id = <u_id> AND uw_i_id = i_id AND uw_i_u_id = i_u_id
ORDER BY i_end_date ASC LIMIT 25
```