openBarter

"the greatest wealth for the lowest collective effort"

Abstract

openBarter is a server based on the database postgreSql implementing a barter engine that produces movements from barter orders submitted.

Author

Olivier Chaussavoine, Project initiator and leader of openBarter

olivier.chaussavoine@gmail.com

1 Introduction

Use of money is widespread due to low liquidity of bilateral barter. If you want to provide a value in exchange of an other it is unlikely that you find someone that wants to provide the value you are looking for in exchange of the one you provide. This “double coincidence of wants problem” is simply solved using money. It is no more a problem when non bilateral exchange are also considered.

In practice, the number of possible exchange combinations grows for markets where the diversity of the kinds of values exchanged is low. Typical examples are raw materials markets and any markets exchanging most vital resources. These values are fungible since they can be measured by a quantity using a physical standard (for example Kg). This is the case even for green house gases, radioactive pollution or enegry.

A regular market implements the best price rule. It is the lowest price for the buyer and the highest for the seller. Among possible relations between unmatched orders of the market, this rule determines the bilateral cycle between a buyer and a seller chosen to form two movements. The first is one where the buyer provides money to the seller. The second is one where the seller provides some good – also a value - to the buyer. Both movements make the cycle and the transaction.

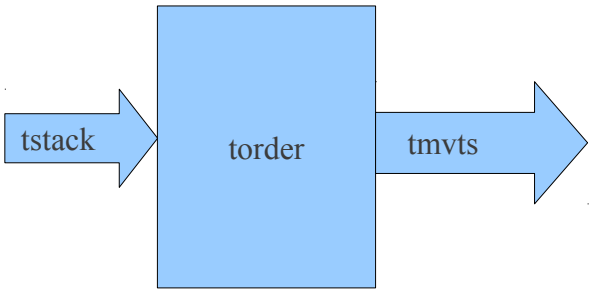
openBarter represents fungible values by a couple (quality, quantity) where the quality is a name describing a quality standard and where the quantity is an integer. It extends the central limit order book mechanism by exploring non-bilateral cycles and produces transactions with more than two movements. It does not require the expression of prices to preform competition that is equivalent to the best price rule when it is applied to bilateral cycles where money is one of the fungible values exchanged.

The value of a resource is simply measured by it's quantity without any reference to a currency standard. For a given quality, the value is proportional to the quantity. The proportionality is limited by boundaries defining different markets such as gallon and barrel for petrol in England. But values with distinct qualities can only be compared subjectively by the market.

2 Architecture

The market is seen as a flow where input are orders and output are movements.

Primitives are available to submit orders and consume movements. The production of orders is done by a function launched by a batch.



It uses four tables. The order book stored in a table *torder*, a stack accepting orders as input *tstack* and a stack *tmvt* storing movements to be consumed. A table *towner* is used to store names of owners of the orders of the order book.

3 Installation

3.1 Build from sources

Following instructions has been tested on linux 32 bits and 64 bits architecture with version 9.2 of postgreSql.

Follow instructions of postgreSql manual to install the sources of the database.

In the contrib/ directory of the sources of postgreSql, install the sources of openbarter using the package you downloaded from github:

```
$ cd contrib
$ gunzip olivierch-openBarter-vx.y.z.tar.gz
$ tar xf olivierch-openBarter-vx.y.z.tar
```

the package is compiled with:

```
$ cd openBarter/src
$ make
$ make install
```

3.2 Tests

To run tests, cd to openBarter/src and:

```
$ make installcheck
…
============== running regression test queries        ==============
test testflow_1               ... ok
...
test testflow_n               ... ok
============== shutting down postmaster               ==============

======================
 All n tests passed.
======================
```

3.3 Install the model

When the postgreSql server is running, the model can be installed. It is defined by the file openBarter/src/sql/model.sql. You must connect with the superuser role *postgres* used for installation of the database, but never user for market operations. When you are in openBarter/src:

```
$ createdb -Upostgres market
$ psql -Upostgres market
psql (9.2.0)
Type "help" for help.

market=# \i sql/model.sql
…..
```

The model does not depend of any schema, and creates some roles if they do not exist yet. You quit psql by typing ctr-D.

4 Use cases

4.1 Non bilateral exchange cycle

If you start a client of postgres, you can obtain the version of the model with the command:

```
$ psql -Upostgres market
psql (9.2.0)
Type "help" for help.

market=# select * from fversion();
         fversion
--------------------------
 openBarter VERSION-X.Y.Z
(1 row)
```

We consider three partners a,b,c where:

- a provides 20 q1 and requires 10 q2
- b provides 20 q2 and requires 10 q3
- c provides 20 q3 and requires 10 q1

The market can find a relation between these orders to for a cycle with partners a, b and c. We insert these orders with the following commands:

```
market=# select * from fsubmitbarter(1,'a',NULL,'q2',10,'q1',20,NULL);
 id | diag
----+------
  1 |    0
(1 row)
market=# select * from fsubmitbarter(1,'b',NULL,'q3',10,'q2',20,NULL);
 id | diag
----+------
  2 |    0
(1 row)
market=# select * from fsubmitbarter(1,'c',NULL,'q1',10,'q3',20,NULL);
```

```
 id | diag
----+------
  3 |    0
(1 row)
```

The diag=0 means the commmand where accepted into the input. The id field returns the number given by the market to the order. Commands are now stacked into the order book contained in the table *tstack.*

A batch should be set to consume this table and submit orders to the order book. When this batch is not installed, you must type the following command:

```
market=# select * from femptystack();
 femptystack
-------------
           3
(1 row)
```

The three orders have been submitted to the order book and produced the following:

```
market=# select id,nbc,grp,own_src,own_dst,qtt,nat from tmvt;
 id | nbc | grp | own_src | own_dst | qtt | nat
----+-----+-----+---------+---------+-----+-----
  1 |   3 |   1 | c       | b       |  20 | q3
  2 |   3 |   1 | b       | a       |  20 | q2
  3 |   3 |   1 | a       | c       |  20 | q1
(3 rows)
```

Three movements have been created where the partner:

- c provided 20 q3 to b
- b provided 20 q2 to a
- a provided 20 q1 to c

A movement is inserted with the name of the database user that inserted the order.

```
market=# select usr,ack from tmvt order by id asc limit 1;
   usr    | ack
----------+-----
 postgres | f
(1 row)
```

The function *fackmvt()* is used to acknowledge the oldest movement:

```
market=# select fackmvt();
 fackmvt
---------
       1
(1 row)

# select usr,ack from tmvt order by id asc limit 1;
   usr    | ack
----------+-----
 postgres | t
(1 row)
```

The *ack* flag is set when a movement is acknowledged. An exchange cycle is removed from *tmvt* when all its movements were acknowledged:

```
market=# select fackmvt();
 fackmvt
---------
```

```
       1
(1 row)
market=# select fackmvt();
 fackmvt
---------
       1
(1 row)
market=# select usr,ack from tmvt;
 usr | ack
-----+-----
(0 rows)
```

4.2 Several barter orders for the same value offered

Several orders of the same owner can be made on a single value offered. It is done by using as third parameter of *fsubmitorder* the reference of an order previously inserted that was offering this value.

We suppose for example that:

- a provides 10 q1 and requires 10 q2 **or** 10q3
- b provides 5q2 and requires 5q1
- c provides 5q3 and requires 5q1

```
market=# select * from fsubmitbarter(1,'a',NULL,'q2',10,'q1',10,NULL);
 id | diag
----+------
  4 |    0
(1 row)
market=# select * from fsubmitbarter(1,'a',4,'q3',10,NULL,NULL,NULL);
 id | diag
----+------
  5 |    0
market=# select * from fsubmitbarter(1,'b',NULL,'q1',5,'q2',5,NULL);
 id | diag
----+------
  6 |    0
market=# select femptystack();
-------------
           3
(1 row)
```

Two movements were produced where a and b exchanged 5q2 for 5q1. These movements are acknowledged:

```
market=# select id,nbc,grp,own_src,own_dst,qtt,nat from tmvt;
 id | nbc | grp | own_src | own_dst | qtt | nat
----+-----+-----+---------+---------+-----+-----
  4 |   2 |   4 | b       | a       |   5 | q2
  5 |   2 |   4 | a       | b       |   5 | q1
(2 rows)
market=# select fackmvt();
 fackmvt
```

```
---------
       1
(1 row)
market=# select fackmvt();
 fackmvt
---------
       1
(1 row)
```

Two orders remain unmatched in the order book:

```
market=# select id,own,oid,qtt_requ,qua_requ,qtt_prov,qua_prov,qtt from
vorder;
 id | own | oid | qtt_requ | qua_requ | qtt_prov | qua_prov | qtt
----+-----+-----+----------+----------+----------+----------+-----
  5 | a   |   4 |       10 | q3       |       10 | q1       |   5
  4 | a   |   4 |       10 | q2       |       10 | q1       |   5
(2 rows)
```

The quantity of q1 owned by *a* remaining available for exchange is 5.

We insert a new order from *c*:

```
market=# select * from fsubmitbarter(1,'c',NULL,'q1',5,'q3',5);
 id | diag
----+------
  7 |    0
market=# select femptystack();
-------------
           1
```

Two new movements were produced:

```
market=# select id,nbc,grp,own_src,own_dst,qtt,nat from tmvt;
 id | nbc | grp | own_src | own_dst | qtt | nat
----+-----+-----+---------+---------+-----+-----
  6 |   2 |   6 | c       | a       |   5 | q3
  7 |   2 |   6 | a       | c       |   5 | q1
(2 rows)
market=# select fackmvt();
 fackmvt
---------
       1
(1 row)
market=# select fackmvt();
 fackmvt
---------
       1
(1 row)
```

Where the remaining quantity of q1 owned by *a* is exchanged for 5 q3 that was owned by *c*.

The order book is now empty, as tables tmvt and torder.

4.3 BEST and LIMIT barter

An owner defines a ratio ω between offered and required quantities. When a cycle is produced from this order, the ratio ω' between provided and received quantities of movements is not equal to ω, but the result of a barter between partners of the exchange cycle sharing economic product of the exchange between them. An order is LIMIT when it's author requires that $\omega' < \omega$. A barter order is

BEST when this condition is not required.

Cycles produced by the market have Ω >=1 except when all orders of the cycle are BEST. In that particular case, cycles can have Ω <1.

The first parameter of *fsubmitbarter* is 1 for limit and 2 for best. We had barter limit in previous examples.

We suppose that:

- a provides 10 q1 and requires 20 q2 (barter best)
- b provides 10 q2 and requires 20 q1 (barter best)

```
market=# select * from fsubmitbarter(2,'a',NULL,'q2',20,'q1',10);
 id | diag
----+------
  8 |    0
(1 row)
market=# select * from fsubmitbarter(2,'b',NULL,'q1',20,'q2',10);
 id | diag
----+------
  9 |    0
market=# select femptystack();
-------------
           2
(1 row)
```

Two movements were produced where a and b exchanged 10 q2 for 10 q1 even if the ratio of movements is not better than those of orders:

```
market=# select id,nbc,grp,own_src,own_dst,qtt,nat from tmvt;
 id | nbc | grp | own_src | own_dst | qtt | nat
----+-----+-----+---------+---------+-----+-----
  8 |   2 |   8 | b       | a       |  10 | q2
  9 |   2 |   8 | a       | b       |  10 | q1
(2 rows)
market=# select fackmvt();
 fackmvt
---------
       1
(1 row)
market=# select fackmvt();
 fackmvt
---------
       1
(1 row)
```

These orders would not produce any movement if orders were barter limit instead of barter best.

4.4 Quote

A quote gives the best path of the market from a quality to an other.

We submit three barter with two possible exchanges of q1 in exchange of q3 and a quote (unnecessary responses à omitted):

```
market=# select * from fsubmitbarter(1,'a',NULL,'q2',20,'q1',80,NULL);
market=# select * from fsubmitbarter(1,'b',NULL,'q3',10,'q2',20,NULL);
market=# select * from fsubmitbarter(1,'c',NULL,'q3',10,'q1',70,NULL);
market=# select * from fsubmitquote(1,'d','q1','q3');
market=# select * from femptystack();
market=# select json from tmvt;
                                        json
-----------------------------------------------------------------------------------------------
 {"qtt_requ":80,"qtt_prov":10,"qtt":10,"qtt_reci":80,"qtt_give":10,"paths":[
+
 [{"type":1, "id":11, "oid":11, "own":2, "qtt_requ":10, "qtt_prov":20, "qtt":20, "flowr":20},+
 {"type":1, "id":10, "oid":10, "own":1, "qtt_requ":20, "qtt_prov":80, "qtt":80, "flowr":80}, +
 {"type":13, "id":13, "oid":13, "own":4, "qtt_requ":80, "qtt_prov":10, "qtt":10, "flowr":10}]+
 ]}
(1 row)
market=# select fackmvt();
market=# select fsubmitbarter(1,'d',NULL,'q1',80,'q3',10,NULL);
market=# select * from femptystack();
market=# select id,nbc,grp,own_src,own_dst,qtt,nat from tmvt;
 id | nbc | grp | own_src | own_dst | qtt | nat
----+-----+-----+---------+---------+-----+-----
 11 |   3 |  11 | d       | b       |  10 | q3
 12 |   3 |  11 | b       | a       |  20 | q2
 13 |   3 |  11 | a       | d       |  80 | q1
(3 rows)
market=# select fackmvt();select fackmvt();select fackmvt();
```

A single row is produced by *fsubmitquote()* with a field *json* of the table *tmvt* giving *qtt_requ* and *qtt_prov* that can be obtained with the best path of the market, and the details of the cycle found. This single row does not represent an exchange, but the result of the quote.

A barter of these quantities give the expected cycle.

5 Principle

The barter market accepts exchange orders of the form:

I offer a value in exchange of an other value of an other quality

submitted by the owner of the value offered.

The market finds potential exchange cycles with two partners or more. Agreements can be formed from them, defined by a set of movements where each partner provides a quantity to an other and receives at the same time a value of an other quality. By allowing more than two partners the liquidity of the market is not limited by the double coincidence of wants problem.

For an order, we define ω as a ratio between provided quantity and minimum required quantity. This ratio measures the will to exchange as would do a price but without the need of a currency. The dimension of ω depends on qualities exchanges and compare such values would not make any sense when these qualities are different. For a cycle formed by several orders where quality offered and required match, we compute an Ω as the product of ω of orders of a cycle. Since Ω a dimensionless quantity, compare these values could have a meaning. When Ω is lower than 1, the collective will to exchange is not sufficient to find an agreement between partners due to minimum quantities required. When Ω is 1 it is easy to find an agreement than match minimum ratio ω required by maximizing the flow of values through the cycle within limits defined by available quantities. When Ω is greater than 1 the excess Ω-1 can be shared fairly for the benefit of partners in order to form the exchange. This share changes ω to a value $\omega' = \omega * \Omega^{-1/n}$ where n is the number of partners so that the product of ω' is 1. The value ω' lower than or equal to ω represents a benefit for the corresponding partner. The share is fair since the ratio ω'/ω is the same for all partners of the cycle.

When an owner submits an order, that's because he considers that the value expected is more useful that the one he owns, and ω measures how much this exchange would be useful for him. For a given cycle Ω is proportional to any ω of it's orders since Ω is their product. In other words Ω is an aggregate common to all partners of the cycle measuring how much the exchange is useful for them, even if usefulness depends on the view point. When a common order belongs to several possible exchange cycles, the author of this order can use Ω as the measurement of the usefulness of potential cycles and compare them to choose the best. For bilateral exchanges, it has be shown[1] that the choice of the cycle having Ω maximum is the same as what would be obtained with the best price rule. In other words, Ω maximization extends the best price rule to non-bilateral exchanges. By maximizing Ω the market meets the goal of utilitarians by maximizing utility but with a definition of utility that would be independent of any currency. Most importantly, it maximizes the collective will to exchange and the common wealth.

The extension of the best price rule to non-bilateral exchange is not unique, and could also be obtained by maximizing individual profit[1] instead of the collective will to exchange. The use of money maintains a confusion between these two distinct goals with assessed social economic consequences.

1 minimize ω', that is minimizing $\Omega^{-1/n}$ or maximizing $\Omega^{1/n}$ instead of Ω .

This market proceeds as a regular market – a central limit order book (CLOB) - by processing orders one after the other. We describe here what is common between these markets. The input of the market is a flow of orders and it's output is a flow of movements. It records unmatched orders in an order book. When a new order is submitted, a competition is performed between potential cycles created by the new order and pending orders in the book to choose the cycle that will form the exchange. If no matching is found, the new order is added to the book. Otherwise, movements forming the exchange are produced from the best cycle and the values offered by matched orders are decreased of the values exchanged. If some cycles remain the competition is repeated as long as the new order is not exhausted.

The difference between this barter market and a regular CLOB is only that 1) exchange cycles can be non-bilateral 2) competition is performed with Ω instead of price.

6 Orders

Orders can be a barter, a quote or remove.

6.1 Barter

For a barter order, the owner offers the value provided in exchange of a value required.

A barter can be LIMIT; meaning that exchange cycles produced from this barter will have a ratio ω' (quantity provided/quantity received) lower or equal to the ratio ω (qua_prov/qua_requ) of the order. A barter is BEST otherwise.

A cycle is LIMIT if at least one barter order of this cycle is LIMIT. For such a cycle, we have ω'<=ω. Otherwise, the cycle is BEST and we can have both ω'>ω and ω'<=ω.

A barter is active for a limited time interval. This delay starts at the time the barter order is inserted into the stack. After this delay,the barter order does not produce exchange cycle.

When a barter expresses a different requirement for a value already offered by a previous barter, it is called a child order. This previous barter order is a parent order. The time interval and quality offered of a child order is that of it's parent but the quality required and ratio ω of the child is distinct from the parent.

6.2 Quote

A quote provides information on possible exchange of the market, but does not change the order book. The result is recorded into the json field of a single movement. This json field contains a dictionnary with keys *qtt,qtt_reci, qtt_give* and *paths*. We note these fields *json.qtt,json.qtt_reci, json.qtt_give and json.paths.*

For a regular market, the result of a quote depends on the quality and quantity bought or sold, and on the price. For openBarter, it depends on the couple (*quality_prov*, *quality_requ*), on the ratio ω (*qtt_prov*/*qtt_requ*) and on the quantity provided *qtt*.

We call path a chain of barter orders of the order book where the first requires the *quality_prov* and the last provides the *quality_requ*, each order of this chain providing the quality required by the next. The general processing is the following:

Fetch paths of the order book for the given (*quality_prov*, *quality_requ*),

For each path, compute Ω as the product of ω of the cycle formed by this path and the barter described by the quote,

For each path ordered by decreasing Ω.

adjust ω to ω' so that their product is 1

compute the flow of value through the cycle

add the path to *json.paths*

add the quantity of *quality_requ* going out of the path to *json.qtt_reci*

add the quantity of *quality_prov* going into of the path to *json.qtt_give*

There are three types of quote. Any type of quote requires the couple (*quality_prov*, *quality_requ*) as input parameters.

A quote (form 3) depends on (*qtt,qtt_prov,qtt_requ,quality_prov*, *quality_requ*). These parameters are sufficient to set a barter order, and the result of the quote is the same as that barter order with a different form. *qtt* is the quantity offered of the *quality_prov* and ω is defined by (*qtt_prov*/*qtt_requ*). (*json.qtt_reci*,*json.qtt_give*) are the quantities of (*quality_prov*, *quality_requ*) that would be exchanged by the barter order with the same parameters.

A quote (form 2) depends on (*qtt_prov,qtt_requ,quality_prov*, *quality_requ*). It is similar to the form 3, but with the parameter *qtt* undefined. The output field *json.qtt* provides this parameter as the maximum flow of value through the cycles to obtains the maximum from the market. A barter order based on *json.qtt* as field *qtt* and other input parameters of this quote would provide the same results as that of the quote.

A *prequote* depends only on (*quality_prov*, *quality_requ*). It provides a first approximation of the quantity going in and out of available paths that would form form cycles. Since the ratio ω of the quote is unknown, it is supposed to be 1 for Ω computation. There is no adjustment of ω to ω' of orders of the path, and the flow of value through the cycle is not computed. A flow through the path is obtain using ω and *qtt* of it's orders, and output are (*json.qtt_reci*,*json.qtt_give*) and *json.paths*. *json.qtt_give*/ *json.qtt_reci* provides an approximation of the ω of the market.

6.3 Remove

A parent barter order can be removed from the order book. Child orders related to this parent are also removed at the same time.

7 Interfaces

Interactions with the order book can be submissions of orders to be executed or administrative tasks. Orders submitted are recorded in a table tstack representing a stack. The stack is consumed to execute orders.

Results of interactions are recorded in the table tmvt. Records of this table represent movements for barter orders, a barter order producing several movements. A record of this table can also represent the result of other interactions.

7.1 Order processing

An order is first submitted and later executed by consuming the stack. It's submission immediately returns a type *yressubmit* with fields *id* and *diag*. *diag=0* means the order was successfully submitted to the input stack. The field *id* gives a unique identifier used later to refer to this order. Execution result(s) in the table *tmvt* have a field *xid* containing the identifier of the barter order.

7.2 Barter order

A barter order can be a parent or a child barter.

A parent barter offers a value in exchange of an other value. A child barter refers to the value offered by a previous parent barter asking for a different value. That way, different values can be asked for the same value offered.

7.2.1 Submission

A parent barter has the form:

```
=> SELECT * FROM
fsubmitbarter(type,own,NULL,qua_requ,qtt_requ,qua_prov,qtt_prov,qtt,dura
tion);
```

A child barter:

```
=> SELECT * FROM
fsubmitbarter(type,own,oid,qua_requ,qtt_requ,NULL,qtt_prov,NULL,NULL);
```

A child barter defines a new requirement for a value already offered by the parent barter *oid*. The fields *qua_prov*,*qtt* and *duration* of a child barter are defined by the parent barter.

Possible error codes returned by *diag* are:

diag	meaning
0	no error. The field id of the response is the number given by the market to the order,

-1	For a parent order, qua_prov and qua_requ must be different
-2	Incorrect parent order
-3	Type must be 1 or 2
-4	Incorrect child order
-5	own must be a non empty string.

The parameters of the function *fsubmitbarter* are the following:

type	int	1 barter limit 2 barter best
own	text	the name of the owner, non empty string.
oid	int	id of the parent order for a child barter. Must be NULL for a parent barter.
qua_requ	text	the quality required, non empty string.
qtt_requ	int8	Defines ω = qtt_prov/qtt_requ with the field qtt_prov ,
qua_prov	text	the quality provided, non empty string. Must be NULL for a child order.
qtt_prov	int8	Defines ω = qtt_prov/qtt_requ with the field qtt_requ ,
qtt	int8	The quantity provided. Must be NULL for a child order.
Duration	duration	Validity delay of the order. It has the type *interval* of postgres, for example: '1 hour'::interval. Must be NULL when *oid* is not NULL.

7.2.2 Execution results

A barter order is stored in the order book when it is executed. It remains in the book until it's duration is reached or the value provided is exhausted. It's presence in the book produces movements in the table tmvt described in §7.5.

Each row of this table represents a statement where an owner *own_src* provides a value (*nat,qtt*) to an other owner *own_dst*. Rows with the same *grp* field define an exchange cycle.

The field *refused* is 0 when the movement describes a cycles.

When a child barter is executed and the parent order is not present in the order book, the field *refused* is -1.

7.2.3 Quotation

As any order, a quote is first submitted to be executed later using the order book. Execution produces a single movement containing the result of the quote, but does not change anything to the order book.

7.2.3.1 fsubmitquote

A quote is submitted with the following syntaxes:

It can be of form 2:

```
=> SELECT * FROM
fsubmitquote(type,own,qua_requ,qtt_requ,qua_prov,qtt_prov,NULL);
```

or form 3:

```
=> SELECT * FROM
fsubmitquote(type,own,qua_requ,qtt_requ,qua_prov,qtt_prov,qtt);
```

Their execution give the result that would be returned by a barter with these parameters. The execution result of form 2 gives the parameter *qtt* that was not defined as input.

The submission response has the type *yressubmit* with the field '*id*' and '*diag*'. Returns diag=0 and an int in the id field. On error, diag contains the code of the error, and id is 0.

When the quote order has no error, it is recorded to be executed. When executed, The result is a recorded as a single movement with a json field. This field is a string representing a dictionary where (*qtt_requ*,*qtt_prov*,*qtt*) are the parameters of the barter command (*qtt_requ*,*qtt_prov*,*qtt*) that would produce this result.

A quote does not insert anything into the order book, but records a dummy movement into the movement table with the informations required.

Possible error codes returned by diag are:

diag	meaning
0	no error. The field id is the number given by the market to the order,
-1	qua_prov and qua_requ must be different
-3	Incorrect order type

A quote submission produces a movement where the *json* field describes the best cycles of the market require-ring and providing the qualities specified. This json string is described in the paragraph 7.2.3.3.

7.2.3.2 fsubmitprequote

A prequote is submitted with the following syntaxes:

```
=> SELECT * FROM fsubmitprequote(own,qua_requ,qua_prov);
```

A prequote submission produces a movement where the *json* field describes the paths of the market require-ring and providing the qualities specified. This json string is described in the paragraph 7.2.3.3.

There is no barter between orders since ω is not defined. It describes the maximum flow of values going through paths.

7.2.3.3 Json returned by a quote or a prequote

The movement produced by a quote or a prequote has a *json* field, a dictionary with the following fields:

qtt_requ	int	Fields *qtt_prov*,*qtt_requ*,*qtt* are what should contain the barter order to obtain the *paths* if this barter was submitted. These fields are 0 for a prequote.
qtt_prov	int	
qtt	int	
qtt_reci	int	quantity received, sum of quantities produced by the paths
qtt_give	int	the quality given, sum of quantities required by the paths
paths	text	The details of paths found

paths is a list of dictionaries representing a path. The dictionary has the following fields:

type	int	The type of the order
id	int	id of the order
oid	int	oid of the order
own	int	owner of the order
qtt_requ	int	defines ω of this order (qtt_prov/qtt_requ)
qtt_prov	int	
qtt	int	Quantity remaining available for this order
flowr	int	quantity of the flow for this order

For example:

```
select * from fsubmitquote(1,'d','q2',10,'q1',10);
 id | diag
----+------
 27 |    0
(1 row)

select json from fproducemvt();
                        json
-------------------------------------------------------------------
{"qtt_requ":10,"qtt_prov":10,"qtt":41,"qtt_reci":60,"qtt_give":41,
"paths":[                    +
 [{"type":1, "id":23, "oid":23, "own":3, "qtt_requ":10, "qtt_prov":30,
"qtt":30, "flowr":30},
{"type":133, "id":27, "oid":27, "own":4, "qtt_requ":10, "qtt_prov":10,
"qtt":17, "flowr":17}],
[{"type":1, "id":22, "oid":22, "own":2, "qtt_requ":10, "qtt_prov":20,
"qtt":20, "flowr":20},
{"type":133, "id":27, "oid":27, "own":4, "qtt_requ":10, "qtt_prov":10,
"qtt":14, "flowr":14}],
[{"type":2, "id":21, "oid":21, "own":1, "qtt_requ":10, "qtt_prov":10,
"qtt":10, "flowr":10},
{"type":133, "id":27, "oid":27, "own":4, "qtt_requ":10, "qtt_prov":10,
"qtt":10, "flowr":10}]]}
```

Represents 3 cycles of orders surrounded by brackets, the last order representing the quote (type&15=133). Quantities flowr of this order are 17,14 and 10, and the sum is 41. The order before the order representing the quote is that providing a flow, the values are 30,20,10 and the sum is 60. qtt_reci and qtt_give are those sums.

Meaning that the order:

```
=> SELECT * FROM fsubmitbarter(1,'d',NULL,'q2',10,'q1',10,41,NULL);
```

would produce movements providing 60 q2 to d in exchange of 41 q1.

7.3 Administrative tasks

These functions should be called regularly.

7.3.1 Consume stack

A function is called to consume the input stack and execute orders. It should be called by a batch as soon as tstack is not empty.

```
=> SELECT * FROM fproducemvt();
```

This function unstack a single order and submit it to the order book.

```
=> SELECT * FROM femtystack();
```

This function unstack all order from *tstack* in a single transaction. The first form should be preferred when the stack is big in order to keep transactions small enough.

7.3.2 Clean outdated barter orders

Outdated barter order cannot belong to cycles founds, but remain in the order book. A function is called to remove them from the order book. It should be called at regular time intervals:

```
=> SELECT * FROM fcleanoutdatedorder();
```

For each parent barter order, a new movement is inserted with the field *refused*=4 and the id of the parent order is in the field *xid*. Child barter orders of these parents are removed at the same time.

7.3.3 Clean old owners

The table *towner* is used to represent owners using as an integer in the order book. This table must be cleaned regularly by the following command:

```
=> SELECT * FROM fcleanowners();
```

It should be called periodically.

7.4 Read the order book

The order book can be red with the following select:

```
=> SELECT * FROM vorder o WHERE o.qua_prov ='gold' DESC LIMIT 10
```

The parameters in bold give the quality provided.

The columns returns are the following:

id	int	Serial number of the order
own	text	Author of the order
oid	int	Referenced order
qtt_requ	int8	Quantity required
qua_requ	text	Quality required
qtt_prov	int8	Quantity provided
qua_prov	text	Quality provided
qtt	int8	Quantity provided remaining for barter
created	datetime	When the order was submitted

Qtt is the quantity available for exchange while qtt_prov and qtt_requ defined the ω of the order. Qtt is reduced each time a movement is created from this order.

When oid is not NULL, the fields qtt,qtt_prov are those of the order referenced by oid.

7.5 Table tmvt

A row of the table tmvt contain movements resulting of barter, or other order results. It has the following fields:

id	int	Serial number of the movement
type	int	order type of the order origin of this movement.
json	text	Information provided by quotes, info or error message.
nbc	int	number of movements in the cycle
nbt	int	number of movements in the transactions
grp	int	id of the first movement of the cycle
xid	int	id of the order origin of this movement
usr	text	database user that inserted the order
xoid	int	Parent of the order origin of this movement
own_src	text	owner providing the value
own_dst	text	owner receiving the value
qtt	int8	quantity moved
nat	text	quality moved
ack	boolean	movement acknowledged (boolean)
exhausted	boolean	quantity of the order exhausted (boolean)
refused	int	Error code of the event: 5 rmbarter order: executed 4 outdated barter removed from the order book 1 quote or prequote executed 0 the movement describes a cycle -1 barter order: the parent order was not found in the order book -2 barter order: owner of order and parent are different -3 barter order: the parent have a parent order

		-4 barter order: the order is too old -7 rmbarter order: failed
order_created	datetime	submission date of the order origin of this movement.
created	datetime	Date of the transaction; when the movement was inserted.

The oldest movement can be accepted with the command:

```
=> SELECT * FROM fackmvt();
```

A movement is accepted by the database user that submitted the corresponding order. All movements with the same (grp) field are removed when they are all accepted.

7.6 Roles

The following roles are defined by openBarter and should not be used for other purposes:

role_co, role_client,role_bo,role_batch

The users must inherit from the role role_client to submit an order, acknowledge a movement or read tables. A super user can disabled/enabled access of users with the command:

```
=> REVOQUE ROLE role_co FROM role_client;
=> GRANT ROLE role_co TO role_client;
```

A single user role_batch is allowed to execute batch functions. A super user can disabled/enabled access of users with the command:

```
=> REVOQUE ROLE role_bo FROM role_batch;
=> GRANT ROLE role_bo TO role_batch;
```

8 Parameters

Parameters of the model are the following:

MAXCYLE	64	maximum number of partners of a cycle. This value can be at maximum 64.
MAXPATHFETCHED	1024	maximum number of cycles on witch competition occurs
MAXMVTPERTRANS	128	Maximum number of movements produced by a single transaction.

MAXCYCLE and MAXPATHFETCHED determine the breadth and depth of the exploration of combination of matching between orders. The default values can be changed by a super user while the model is running. By increasing these values, the fluidity of the market grows, and the computation time to process orders increases.

A single transaction can record many cycles, and MAXMVTPERTRANS is used to limit the volume of data of the transaction. When this limit is reached, the transaction is committed without any accounting errors.

9 Installation

9.1 Build from sources

Following instructions has been tested on linux 32 bits and 64 bits architecture with version 9.2 of postgreSql.

Follow instructions of postgreSql manual to install the sources of the database.

In the contrib/ directory of the sources of postgreSql, install the sources of openbarter using the package you downloaded from github:

```
$ cd contrib
$ gunzip olivierch-openBarter-vx.y.z.tar.gz
$ tar xf olivierch-openBarter-vx.y.z.tar
```

the package is compiled with:

```
$ cd openBarter/src
$ make
$ make install
```

9.2 Tests

To run tests, cd to openBarter/src and:

```
$ make installcheck
…
============== running regression test queries        ==============
test testflow_1               ... ok
...
test testflow_n               ... ok
============== shutting down postmaster               ==============

======================
 All n tests passed.
======================
```

9.3 Install the model

When the postgreSql server is running, the model can be installed. It is defined by the file openBarter/src/sql/model.sql. You must connect with a superuser role that is never user for market operations. When you are in openBarter/src:

```
$ createdb  -Upostgres market
$ psql  -Upostgres market
psql (9.2.0)
Type "help" for help.
```

```
market=# \i sql/model.sql
…..
```

The model does not depend of any schema, and creates roles *client* and *admin* if they do not exist yet. You quit psql by typing ctr-D.

9.4 Releases

0.1.0

First release. Tests units are functional [Olivier Chaussavoine].

0.1.1

Berkeley-db is resides in memory instead of files in $PGDATA. This increases global performance of searches. [Olivier Chaussavoine]

0.1.2

rights of roles of the database model are defined globally using schemas instead of granted individually for each function. [Olivier Chaussavoine]

0.1.6

ported on postgres9.1.0

0.2.0

The use of berkeleydb is replaced by WITH .. SELECT of PostgreSQL. A new type “flow” is defined, containing low level calculations. Tests units are functional [Olivier Chaussavoine].

0.2.1

Memory allocation and code cleaned. Tests units are functional [Olivier Chaussavoine].

0.2.2

Core algorithms optimized. Tests units are functional [Olivier Chaussavoine].

ob_fget_omegas(np,nr) provides the list of all prices found, even those not requested. [Olivier Chaussavoine]

0.3.0

The constraint of acyclic graph is removed. Complete redesign. [Olivier Chaussavoine].

0.4.0

quote and prequote added. [Olivier Chaussavoine].

Order rejection mechanism added [Olivier Chaussavoine].

0.4.1

ported on postgreSql 9.2. [Olivier Chaussavoine].

Bug fixes [Olivier Chaussavoine].

0.4.2

Bug fixes [Olivier Chaussavoine].

0.5.0

fgeterrs() optimized, it can be run when the market is running,

index optimization in fcreate_temp()

increasing preformance of fgetprequote(),fgetquote(),fexecquote(),finsertorder()

X6 faster

MAXCYCLE was 8, it can now be up to 64 [Olivier Chaussavoine].

0.6.0

New model with only 4 tables [Olivier Chaussavoine].

0.6.1

Bug fixes [Olivier Chaussavoine].

0.7.0

barter limit, barter best and quote added [Olivier Chaussavoine].

Schema removed.

0.7.1

new forms of quote,

validity delay added to barter orders.

0.8.0

new unit tests,

validity delay added to barter orders.

improvement of roundings of so that barter and quote produce exactly the same result.

0.8.1

old quote form 1 removed,

fcleanowners() command added.