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 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. But the problem should disappear 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 surface of forest.

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.

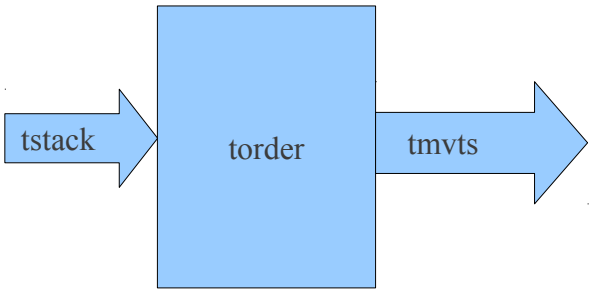
openBarter represents fungible values by a couple (quality, quantity) where the quality is a name describing a quality standard and where the quantity is an integer. It extends the central limit order book mechanism by exploring non-bilateral cycles and produces transactions with more than two movements.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

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

```
 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 table *tstack*.

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

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

The three commands 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 orders for the same value provided

Several orders of the same owner can be made on a single value. It is done by using as third parameter of *fsubmitorder* the reference of a previous order providing 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);
 id | diag
----+------
  4 |    0
(1 row)
market=# select * from fsubmitbarter(1,'a',4,'q3',10,NULL,NULL);
 id | diag
----+------
  5 |    0
market=# select * from fsubmitbarter(1,'b',NULL,'q1',5,'q2',5);
 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

A barter order is limit when the ratio ω' between provided and received quantities of movements produced is better (lower) than the ratio ω between provided and required quantities of the order.

A barter order is best when this condition is not required. The best ω' of the market is used to produce movements even if it is worse (higher) than the ω of order.

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);
market=# select * from fsubmitbarter(1,'b',NULL,'q3',10,'q2',20);
market=# select * from fsubmitbarter(1,'c',NULL,'q3',10,'q1',70);
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,"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);
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 propose a value in exchange of a value of an other quality for a minimum quantity

submitted by the owner of the value proposed.

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. 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 common 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. 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 when this choice is applied to bilateral exchanges. 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 is independent of any currency.

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 utility. The use of money maintains a confusion between these two distinct goals with assessed social economic consequences.

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. CLOB algorithm used by most important market places such as the New York Stock Exchange is adapted to implement barter markets.

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

6 Interfaces

Interfaces allow insertion of new orders into the input stack and consumption of movements produced. A batch performs regular submission of orders using the stack as input. The results of this batch are inserted into the movements table tmvt.

Orders can be a barter or a quote.

For a barter, 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 (quantity provided/quantity received) lower or equal to the ratio (qua_prov/qua_requ) of the order.

A barter can also be best; meaning that exchange cycles produced will be the best possible exchanges of the order book without a limitation on the ratio of the order.

The value offered by a barter can be the same as that offered by a previous parent order.

A quote just provides information on possible exchange of the market. The result is recorded as a single dummy movement containing informations on the best path of the market for the qualities that would be required and provided.

6.1 Input

6.1.1 barter

A barter is submitted with the following syntax:

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

Where:

type	int	1 barter limit 2 barter best
own	text	the name of the owner,
oid	int	id of a parent order,
qua_requ	text	the quality required,
qtt_requ	int8	the quantity required,
qua_prov	text	the quality provided.
qtt_prov	int8	the quantity provided,

Duration	duration	Validity delay of the order.

The response has the type yressubmit with the field 'id' and 'diag'. Returns diag=0 and an int in the id field. This id is the primary key given by the market to this order that will be referred later in other orders or movements. On error, diag contains the code of the error, and id is 0.

oid is set NULL when this barter is a parent order. It is set to the id of a parent order to express the fact the present barter request an other value for the value proposed by the parent order. The fields qua_prov must be the same as parent. The field qtt_prov is ignored. This order is then a new requirement on the value provided by this previous order.

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
-2	qtt_prov<=0 or qtt_requ<=0
-3	Incorrect order type
-4	When oid is not NULL, qua_prov and qtt_prov must be NULL

6.1.2 quote

For a regular market, the result of a quote depends on the quality and quantity of the value quoted. It depends here on the couple (quality provided, quality required), on the ratio (qtt_prov/qtt_requ) and on qtt, the quantity provided. For a given couple (quality provided, quality required), a *prequote* gives the paths requiring and providing theses qualities, while a *quote* gives the result of a barter order.

A *prequote* just returns paths, while a *quote* returns parameters (qtt_prov,qtt_requ,qtt) and the result that would be obtained if a barter order was submitted with these parameters. When the *quote* is not submitted with the three parameters (qtt_prov,qtt_requ,qtt), the result of the quote define those that where not defined by the submission.

6.1.2.1 prequote

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 have the following fields:

qtt_requ,qtt_prov,qtt	int,int,int	These fields are 0
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
path	text	The details of paths found

6.1.2.2 quote

A quote is submitted with the following syntaxes:

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

or:

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

or:

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

These forms give approximations of the result, with a precision growing as the number of parameters submitted. The last form gives exactly the result that would be obtained at that time with a barter order with the same parameters.

The parameters are the same as a barter order.

The response has the type yressubmit with the field 'id' and 'diag'. Returns diag=0 and an int in the id field. This id is the primary key given by the market to this order that will be referred later in other orders or movements. On error, diag contains the code of the error, and id is 0.

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 have the following fields:

qtt_requ,qtt_prov,qtt	int,int,int	The parameters of an order that produce the following result
qtt_reci	int	Quantity received, sum of quantities produced by the cycles
qtt_give	int	Quantity given, sum of quantities required by the cycles
path	text	The details of cycles produced

6.2 Batch

The function that can be called to consume the input stack is the following:

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

This function unstack a single order.

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

This function unstack all order of *tstack* in a single transaction.

6.3 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.

6.4 Output

The table of movements can be read with a SELECT statement.

This table contains results of barter. Each row is a statement where an owner (own_src) provides a value (nat,qtt) to an other (own_dst).

An exchange cycle produced by a barter is described by rows with the same (grp) field.

A single row is produced with (own_src=own_dst) for a quote or when an order was rejected by the batch.

A row of the table tmvt has the following fields:

id	int	Serial number of the movement
type	int	Field defining the order type: type & 3 : 1=LIMIT,2=BEST type & (64|128): 0=barter ,64=prequote,128=quote
json	text	Information provided by quotes or an 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	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 when order is refused: 0 no error -1 the parent order was not found in the order book -2 owner of order and parent are different -3 the parent have a parent order When refused !=0 , then then nbc = 1 and nbt = 1 The json field gives a text for this error.
order_created	datetime	date submission of the parent order if there is one or of the order otherwise.
created	datetime	Date of the transaction.

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.

6.5 Roles

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;
```

7 Parameters

Parameters of the model are the following:

MAXCYLE	16	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 liquidity of the market grows, and the computation time to process orders decreases.

A single transaction can record many cycles, and MAXMVTPERTRANS is used to limit the volume of data of the transaction.

8 Installation

8.1 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
```

8.2 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.
======================
```

8.3 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.

8.4 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.