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.

Authors

Olivier Chaussavoine, Project leader of openBarter

olivier.chaussavoine@gmail.com

1 Introduction

Use of money is widespread due to low liquidity of barter. This low liquidity can be explained by the fact it is unlikely that someone finds someone else that accepts the value he provides and at the same time has the value he requests and accepts to provide it in exchange. This is known as the “double coincidence of wants problem” that is simply solved using money. openBarter extends bilateral exchange between a buyer and a seller to exchange cycles with possibly more than two partners.

Most vital resources can be represented by a couple (quality, quantity) where the quality is a name describing a quality standard and where the quantity is an integer. All such fungible values can be represented this way, including green house gases, radioactive pollution, surface of forest or even money. 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.

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 pending orders of the order book 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 goods – also a value - to the buyer. Both movements makes the cycle and the transaction. openBarter provides an other method to determine the cycle chosen to form movements. This method does not require the expression of prices, but is equivalent to the best price rule when it is applied to bilateral cycles where money is one of the fungible values exchanges.

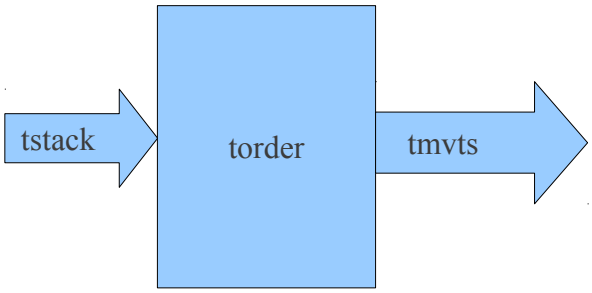
But some goods cannot be represented as fungible values. In this case, the quality cannot be represented by a single name, but requires a text to be described. 'full text search' function of postgreSql has been developed to store documents and retrieve them using logical combination of keywords just like google. This function represents indexed document as tsvector objects, and requests as tsquery. Since the value provided is a real good that can be described using a text and keywords, openBarter use a tsquery to represent it instead of a single name. The request component of the order is expressed as a tsquery instead of a single name. openBarter offers most 'full text search' functions of postgreSql to determine the matching between orders.

Since transports induced by exchange can be significant, openBarter include in orders expressed by participants the place where the value is available, the place where the value required in exchange is expected and a distance. These geographic informations contribute to determine the matching between two orders.

2 Architecture

openBarter use 3 tables in postgreSql. The order book stored in a table torder, a stack accepting orders as input and a stack storing movements to be consumed.

The production of orders is done by a function launched by an external script.



3 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 could be adapted to implement barter markets.

4 Interfaces

4.1 Input

An order is be submitted with the following syntax:

```
=> SELECT * FROM
fsubmitorder(own,oid,quar,qttr,posr,quap,qttp,posp,dist,weight);
```

Where:

own	text	the name of the owner,
oid	int	NULL or id of a parent order,When not NULL fields quap, posp, weight and dist are ignored.
quar	text	the quality required,
qttr	int8	the quantity required,
quap	text	the quality provided. Ignored when oid is NULL.

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

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 usually NULL, but can be set to the id of a parent order to express the fact the present order request an other value for the value proposed by the parent order. The fields quap must be the same as parent. The fields quap is ignored. This order is then a new requirement on the value provided by this previous order.

A parent order must not have parent.

4.2 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.

4.3 Read the order book

The order book can be red with the following select:

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

The parameters in bold give the quality required.

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 remaining
created	datetime	When the order was submitted

Qtt is the quantity available for exchange while qtt_prov is the quantity defined by 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.

4.4 Output

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

id	int	Serial number of the movement
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 ordrer origin of this movement
own_src	text	owner providing the value
own_dst	text	owner receiving the value
qtt	int8	quantity
nat	text	quality
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)
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 of a cycle are removed when they are all accepted.

4.5 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;
```

5 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

MAXCYCLE and MAXPATHFETCHED determine the 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.

6 Installation

6.1 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
```

6.2 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.
======================
```

6.3 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 market
$ psql 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.

6.4 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 [Olivier Chaussavoine].

0.6.1

Bug fixes [Olivier Chaussavoine].

.