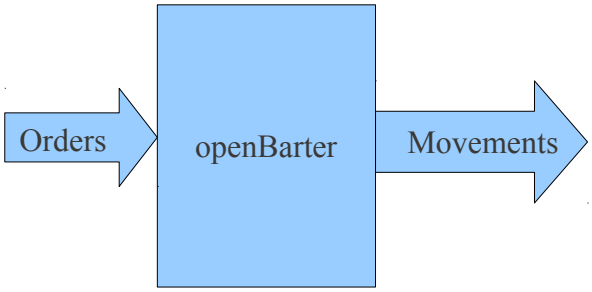
openBarter

openBarter implements a barter market place organized as a *central limit order book* allowing exchange with more than two partners in a single transaction. It is a matching engine accepting exchange orders of owners, and providing movements changing ownership of values according to orders and market rules. It is an extension of postgreSQL.



1 The market rules

An order is expressed by the owner as:

I want to provide a value in exchange of an other value

By value, we mean a couple (quality,quantity) where quality is a name and quantity is an integer.

The quantity of the value provided is owned by the author of the order, and the quantity of the value required is the minimum expected in exchange. Obviously, the quality provided must be different from the quality required.

Two orders are related when the quality provided by the first equals the one required by the other. When related orders define a single cycle, these orders are matched, producing an agreement recorded by a single transaction a set of movements.

Exchange agreements are formed when the ratio (quantity provided/quantity required) produced by the agreement is smaller than that defined by the order. This rule extends the minimum limit of the order to the case where the quantity provided by the agreement is not that of the order.

When several exchange comply with this constraint, the best (minimum ratio quantity provided/quantity required) is chosen for the final agreement.

Due to the number of possible combinations, two limits has been set:

- the maximum number of partners of an agreement is MAXCYCLE (<8).
- the number of agreements on which the competition is performed is limited to MAXORDERFETCH (by default 100).

Even with these limits, openBarter remains far more efficient than what could be done by hand without any computational assistance.

Among all possible agreements, the second limit explores agreements having the smallest number of partners.

2 The model

Let **ω** be the a ratio (quantity provided/quantity required) defined by an order. It measures the pain to give an amount of the quality provided compared to the pleasure to receive a unit of the quality required. The dimension of this measurement is (quality provided/quality required).

For a cycle of orders, let be Ω the product of their **ω**. This product is non dimensional.

When Ω equals to 1, an agreement can be formed where each partner provides some value to an other.

When $\Omega \neq 1$, ω are divided by the geometric mean of ω of the cycle. This division converts ω to ω' in such a way that the product of ω' equals to 1 . This adjustment is a bartering. It is fair when all partners are distinct. When it is not the case, the fairness is maintained by sharing first between partners, then sharing between orders of each partner.

To satisfy the minimum quantity required by the order, we must have $\omega > \omega'$, that is $\Omega > 1$. Otherwise, the cycle is ignored.

When an order forms several cycles, a competition is performed between them by choosing the one having the maximum Ω. This rule applied to cycles formed by two orders is equivalent to the best price rule of a regular market.

3 Implementation

The market is seen as a directed graph where orders define nodes, and relations between orders define arrows. This graph is used to transform orders into agreements when cycles appear on this graph. This can occurs each time an order is added. A competition also occurs between possible cycles when more than one cycles are found. Quantities corresponding to an agreement reduce quantities provided by orders.

The main time consuming primitives of the server are:

- make a quote,
- make an order.

openBarter is an extension of postgreSQL. Stored procedures act on a model representing qualities, owners, orders and movements.

3.1 Database model

The database is described by src/sql/model.sql. It consists in related tables, stored procedures and a special type called “flow” representing a draft agreement that performs fast calculations in C language.

3.1.1 Order book

A table torder represents the order book. When an order is inserted in this table and a cycle is found with $\Omega \geq 1$ movements are created are inserted in a table tmvt, decreasing the quantity provided by the order.

An order is be moved out from this book when it is empty (the provided quantity is 0) to an other table torderremoved. This keeps the order book as small as possible for performance.

3.1.2 Market opening and closing

The life cycle of the market is represented as follows:

A client can use the market during the service phase. Clients must remove all remaining movements and orders at the closure phase. The preparation phase can start at this condition. Then tables are truncated, ready and for a new service cycle.

The service phase stops and closure starts when admin executes:

```
SELECT fclose();
```

All remaining orders are converted to dummy movements (owner to the same owner).

The closure phase stops and preparation starts when admin user executes:

```
SELECT fprepare();
```

This function can be used only when the table of movements tmvt is empty.

The phase preparation stops and service starts when admin executes:

```
SELECT fopen();
```

The market is opened when the market is intialized.

3.1.3 Users

PostgreSQL implements an extensive set of security mechanisms including authentification and access rules. All objects of openbarter can only be modified through predefined functions. A single user “admin” is allowed to perform administration tasks. A role “client” groups the rest of the database users. Clients can get a quote and set an order only during the service phase. They can remove movements and orders only during service and closure phase.

The admin can register a new client, open or close the market, but cannot participate to the market.

The super user that creates the database is distinct from admin and clients. This super user must be used only for this purpose.

3.1.4 Objects

All objects are stored in a schema of the database having the name 't'.

Table	Description
tmvt	History of movements where the ownership of values are modified. It is the output flow of the market.
tmvtremoved	Movement removed
torder	Orders, value provided, value required and owner
towner	Owners
tquality	Qualities
torderremoved	Orders removed
tuser	Description of clients
tmarket	History of market

3.1.4.1 Owner

Users should not be confused with owners. Owners are the authors of orders, while users connect to the database and act on the market on the behalf of owners.

3.1.4.2 Quality

A quality is a string of the form <client_name>/<quality_name>, and belongs to a single client. For example “bankofbali/coconuts” belong to “bankofbali”.

The quality provided by an order must belong to the client that insert it.

Likewise, a client can only remove movements whose quality belongs to him.

A value belongs to an owner while a quality belongs to a client.

3.1.4.3 Movement

Agreement are formed by a set of movements where each partner provides a value he owns to an other partner. An agreement is simply a set of successive record in the table tmvt, where each defines the value, the provider, the receiver (see §3.2.9 for details).

3.2 Application programming interface

The “market” role acts through stored procedures that must be integrated in transactions in the read-commited mode that is the default mode of postgreSQL.

The following list presents functions and views. Some low level routines are written in C language, integrated in a special type “flow”.

Function and views	action	Defined in	Roles allowed
finsertorder	Inserts an order	sql/order.sql	market
fgetquote	Gets a quote	sql/quote.sql	market
fremoveorder	Removes an order	sql/order.sql	market
fgetstats	Produces statistics	sql/stat.sql	admin
vorder	List of orders	sql/init.sql	
vorderremoved	List of removed orders	sql/init.sql	
vmvt	List of movements	sql/init.sql	
vmarket	Market status	sql/admin.sql	
fopen	Open the market	sql/admin.sql	admin
fprepare	Prepare market opening	sql/admin.sql	admin
fclose	Close the market	sql/admin.sql	admin
fcreateuser	Creates a user	sql/admin.sql	admin
fremoveagreement	Remove agreement	sql/user.sql	market

In case of error, an exception is raised depending on it's type.

Error codes	Type
YA001	Quantity of a given quality overflows
YA002	accounting error
YA003	internal error
YU001	abort dues to incorrect use of a primitive

In the following, int is used for 32 bit integer, and int8 for 64 bits integer.

3.2.1 finsertorder

```
SELECT finsertorder(
_owner text,
_qualityprovided text,
```

```
_qttprovided int8,
_qttrequired int8,
_qualityrequired text);
```

conditions :

- ***_qttprovided*** > 0
- ***_qttrequired*** > 0

the function inserts an order made by _owner providing the value (_qttprovided,_qualityprovided) in exchange of a a value having the _qualityrequired and for a minimum quantity of _qttrequired.

The order is inserted in the order book, and possible cycles are found.

Returns a list of records (_uuid text,_dim int,_grp int) representing agreement produced. _uuid is a unique label given by the market to an order and common to all records. _dim is the number of partners of the agreement, and _grp is the reference grouping movements representing the agreement in the tmvt table.

3.2.2 fgetquote

```
SELECT fgetquote (_owner text,_qualityprovided text,_qualityrequired text);
```

conditions :

- ***_qualityprovided*** and ***_qualityrequired*** has been already recorded by previous orders,

Gets a quote for the given couple (_qualityprovided, _qualityrequired) and or the owner _owner.

The quote depends on the owner because fairness of the bartering depends on it (see §2).

Returns records (_dim,_quantityprovided,_quantityrequired), ordered by increasing ratio _quantityprovided/_quantityrequired. _dim is the number of partners of the exchange. Each record represents a cyclec that would be produced if an order with such qualities was submitted. The first ratio has the best ratio.

If for example fgetomega('q/copper','q/iron') returns: (3,5,8),(4,9,10)

then an order finsertorder('me','q/copper',5,8,'q/iron') would produce an agreement with 3 patners exchanging 5 'copper' for 8 'iron'.

and an order finsertorder('me','q/copper',9,10,'q/iron') submitted immediately after would exchange 9 'copper' for 10 'iron'.

It also mean that a total quantity of 8+10 of 'iron' is available on the market for exchange with 'copper'.

3.2.3 fremoveorder

```
SELECT fremoveorder(_uuid text)
```

conditions:

- an order with the label ***_uuid*** exists

The order is removed from the order book.

Returns a row representing the order just removed, as the view ***vorder*** does.

3.2.4 fremoveagreement

```
SELECT fremoveagreement(_grp int)
```

conditions :

- an agreement ***_grp*** exists

The function is called by a client when all movements of the exchange are red from the table of movements. It moves movements of this exchange belonging to this client to the table ***tmvtremoved***. The function returns an integer tha is the number of movements removed.

3.2.5 fcreateuser

```
SELECT fcreateuser(_username text)
```

The function creates the user and provides access to he database with the role client. It can only be executed by admin.

3.2.6 fstats

```
SELECT fstats(_extra bool)
```

gives general informations about the database:

Column	Type
with MAXCYCLE and MAXORDERFETCH	int
Number of qualities	int
Number of owners	int
Number of orders in the order book	int
Number of order removed	int
Number of connections between orders *	int
Number of movements	int
For each _dim, number of agreements with _dim partners *	int
Total number of agreements *	int
Errors on quantities in movements (significant until some movement is removed) *	int
Errors on qualities in movements (significant until some movement is removed) *	int

Lines with (*) are only provided when the _extra parameter is true.

3.2.7 vorder

Gives a description of the order.

Column	Type	Meaning
id	int	Internal reference of the order
uuid	text	External reference of the order
owner	text	Name of the owner
qua_requ	text	Quality required
qtt_requ	int8	Quantity required
qua_requ	text	Quality required
qtt_requ	int8	Quantity required
qtt	int8	Quantity not yet exchanged for this order
created	timestamp	Time when the order was inserted
updated	timestamp	Time when the order was last updated

examples

```
SELECT * FROM vorder WHERE owner='jack';
```

List of orders owned by the owner '***jack***'.

3.2.8 vorderremoved

Same as vorder.

When the quantity left in order is 0, the order is appears in this view. When the order is removed by a client, it also appears here with a qtt >0.

3.2.9 vmvt

It is the list of movements.

Column	Type	Meaning
id	int	Internal id of the movement
nb	int	Number of movements of the agreement
oruuid	text	Reference to the order that produced it
grp	int	id of the agreement. It is the id of the first movement of this agreement.
provider	text	Owner providing the value
nat	text	Quality of the value moved
qtt	int8	Quantity of the value moved

receiver	text	Owner receiving the value
created	timestamp	Time when the agreement was formed

examples:

```
SELECT * FROM vmvt WHERE quality='gold';
```

List of movements of the quality '***gold***'.

3.2.10 vmvtremoved

Same as vmvt but for the table tmvtremoved.

3.3 Installation

3.3.1 Build from sources

Following instructions has been experimented on linux 32 bits and 64 bits architecture.

If you are in the contrib/ directory of postgres, and have unzipped the package into openBarter:

```
>> cd openBarter/src
>> make
>> make install
```

Restart postgres server, and verify test are running:

```
>> make check
…
============== running regression test queries        ==============
test testflow_1             ... ok
test testflow_2             ... ok
test testflow_3             ... ok
test testflow_4             ... ok
============== shutting down postmaster            ==============

======================
 All 4 tests passed.
======================
```

3.3.2 Install the model

The model is defined by the file openBarter/src/sql/model.sql. It is recommended to execute it with a superuser role that is never user for market operations. When you are in openBarter/src:

```
>> createdb market
>> psql market
market=# CREATE SCHEMA t;
market=# SET SCHEMA 't';
market=# \i sql/model.sql
market=# GRANT USAGE ON SCHEMA t TO market,admin;
market=# REVOKE EXECUTE ON ALL FUNCTIONS IN SCHEMA t FROM public;
market=# REVOKE ALL ON ALL TABLES IN SCHEMA t FROM public;
```

The model does not depend of any schema, and creates roles “market”,”client” and “admin” if they do not exist yet, and modify them otherwise.

To start operation, just connect to the database as admin, and create some clients with the function like this:

```
>> psql -Uadmin market
market=> SELECT t.createuser('username')
```

3.4 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]

1.0.0

Complete redesign. [Olivier Chaussavoine].