openBarter

openBarter is an extension of postgreSQL

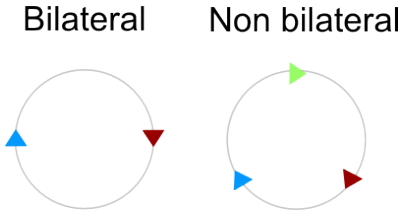
1 Objects of the model

1.1 Value

A *value* is a couple (*quantity,quality*), where *quality* is a name, and *quantity* is an integer. It can be used to define an amount of mineral, of pollution, or of money. The *owner* of such a value can use the market to exchange it for an other quality.

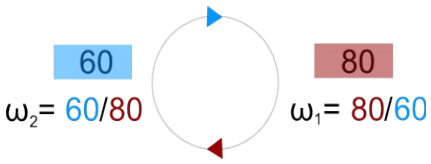
1.2 Exchange

This market allows exchanges between two partners or more in a single transaction. An exchange forms a cycle of partners where each provides a value and receives an other value of different quality. An exchange with more than two partners is called *non-bilateral*.



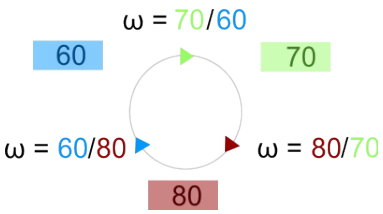
1.3 Price

The *price* is defined for a couple (quality provided,quality required). It is the ratio **ω** between provided quantity and received quantity. It measures how much we accept to provide of the quality provided when we receive a unit of the quality required.
On a regular market, the expression of price is the same for the buyer and the seller and are equal when they agree on the price. In this model, the price of parters are different even when they agree on their prices.



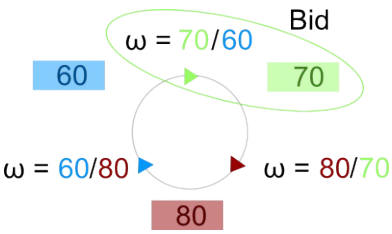
1.4 Agreement on price

When a set of bids form an exchange cycle, a draft agreement can be formed when there is an agreement on prices. It occurs when the product of prices ***ω*** of bids of the exchange cycle equals to ***1***. When this condition is not met a barter is required. The agreement is obtained on the example.



1.5 Bid

It is an unilateral commitment of the owner to exchange a value he owns for another quality at a given price. A bid is defined by a value provided, the quality of the value required, and a price ***ω***. Two bids match when the quality provided by one is the quality required by the other. When some bid matchings form a cycle, an exchange can be formed from this cycle.



1.6 Best price rule

It is used to select the best among possible exchange cycles. The best cycle is the one having the maximum product of prices ω.

1.7 Barter

When there is no agreement on price, an automatic barter is performed where prices 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 rule is fair when all partners are distinct. When it is not the case Ω is first shared between partners, then shared between bids of each partner.

2 Implementation

The market is seen as a directed graph where bids define nodes, and matchings between bids define arrows. This graph is maintained acyclic by transforming cycles into draft agreements as soon as they appear. This occurs each time a bid is added. A competition occurs between possible cycles when more than one cycles are found. Draft agreements are produced from these cycles. Values corresponding to these agreements reduce values of bids in such a way that cycles disappear.

Due to limits of computational resources on time and memory, a limit of 8 has been defined for the maximum number of partners of a draft agreement.

openBarter is an extension of postgreSql. Stored procedures act on a model representing qualities, owners, values, and draft exchange agreements.

The main time consuming primitives of the server are:

- read the best price for a couple of qualities requested and provided,
- make an bid.

2.1 Users

Users of openBarter is a role “market” defined are clients of the database.

2.2 Database model

The database is described by src/sql/model.sql. It consists in related tables, stored procedures and a special type called “flow”.

2.2.1 Vocabulary

quality	The name of a quality.
owner	The name of the owner of a value**.**
value	It is a tuple (quality, quantity,owner).
account	is a position at a given time, with history, for a given (owner,quality)

2.2.2 Schema

Table	Description
ob_tdraft, ob_tcommit	Description of draft agreements. An agreement is a ob_tdraft row. A ob_tcommit row refers to it and describes the commitment of a single owner for this agreement.
ob_tmvt	History of movements on ob_tstock, when the ownership of values are modified.
ob_tnoeud	Description of bids, refers to a quality provided, and a value in ob_tstock. Contains a couple (qtt_prov,qtt_requ) of integers such as ω=qtt_prov/qtt_requ
ob_towner	Description of owners

ob_tquality	Description of qualities
ob_tstock	Description of values

A row of the table ob_tstock describes a value – a tuple (quantity, quality, owner of the value,type). A type that can be A,D or S.

It is a stock_A when the value is moved to the database.

It is a stock_S when the value is referred by a bid.

It is a stock_D when the value is referred by a draft agreement.

2.3 Application programming interface

The "market" role acts through stored procedures that must be integrated in transactions in the read-commited mode.

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

Function and views	action
ob_fadd_account	moves a value to owner's account
ob_fsub_account	moves a value from owner's account
ob_finsert_bid	inserts a bid
ob_finsert_sbid	insert a bid based on an other
ob_fdelete_bid	removes a bid
ob_faccept_draft	accepts a draft
ob_frefuse_draft	refuse a draft
ob_fstats	gives global stats
ob_vowned	Gives quantity owned for each couple (quality,owner).
ob_valance	List of values owned by user group by quality name.
ob_vdraft	List of drafts where the owner is partner.
ob_vbid	List of bids
ob_vmvt	List of movements

In case of error, an exception is raised, with the code "38000", with comments about the error. In the following, int is used form 32 bit integer, and int8 for 64 bits integer.

2.3.1 ob_fadd_account

```
Int ob_fadd_account(_owner text,_quality text,_qtt int8);
```

conditions :

- ***qtt*** >=0.

moves the value to owner's account defined by a couple ***[owner,quality]***

quality and owner are created when they do not exist. The movement is recorded.

Returns:

- 0 when the account is credited,
- <0 on error.

2.3.2 ob_fsub_account

```
int ob_fsub_account(_owner text,_quality text,_qtt int8);
```

conditions :

- ***owner*** and ***quality*** exist,
- ***qtt*** >=0,
- ***qtt*** <= the quantity of the account(***_owner,_quality)***

Moves the value to owner's account defined by the couple ***(_owner,_quality)***

Account is deleted when empty. The movement is recorded.

Returns:

- 0 when the account is debited,
- <0 on error.

2.3.3 ob_finsert_bid

```
int8 ob_finsert_bid(
        _owner text,
        _qualityprovided text,
        _qttprovided int8,
        _qttrequired int8,
        _qualityrequired text
)
```

inserts a new bid based on a new stock_D. Several Drafts can be created by this function that returns the number of draft created.

conditions :

- ***owner*** has an account (_owner,_qualityprovided) with a quantity >= ***qttprovided***,
- ***qualityprovided*** and ***qualityrequired*** are defined,
- ***qttprovided*** >0,
- ***qttrequired*** >0.

Returns:

- >=0 the number of drafts created,
- <0 on error.

2.3.4 ob_finsert_sbid

```
int8  ob_finsert_sbid(
        bid_id int8,
```

```
        qttprovided int8,
        qttrequired int8,
        qualityrequired text
)
```

Inserts a bid based on an other bid. The value proposed by this new bid is the same as the one referred by the bid ***bid_id***.

- ***bid_id*** exists,
- ***qttprovided*** >0,
- ***qttrequired*** >0,
- ***qualityrequired*** is defined.

Returns an int8:

- the number of drafts created (>=0),
- < 0 error.

2.3.5 ob_fdelete_bid

```
ob_fdelete_bid(bid_id int8)
```

conditions :

- the bid exists

Delete bid and related drafts.

Delete related stock_S if it is not related to an other bid. The quantity of this stock_S is moved back to the account of the owner.

A given stock_S is deleted by the ob_fdelete_bid() when it is only referenced by this deleted bid it.

2.3.6 ob_faccept_draft

```
int ob_faccept_draft(draft_id int8,owner text)
```

conditions :

- draft_id exists with status Draft
- owner is partner of this draft

returns:

- 0 the draft is not yet accepted by all partners,
- 1 the draft is executed,
- < 0 error.

2.3.7 ob_frefuse_draft

```
int ob_frefuse_draft(draft_id int8,owner text)
```

conditions :

- draft_id exists with status Draft
- owner is partner of this draft

the draft is cancelled.

Values of stock_D booked for this draft are moved back to stock_S of bids.

returns:

- 0 the draft is cancelled,
- < 0 error.

2.3.8 ob_get_omegas

```
Set of int8[] ob_get_omegas(_nr int8,_np int8)
```

conditions :

- _nr and _np are quality.id

returns a list of couples [qtt_prov,qtt_requ] ordered by qtt_prov/qtt_requ representing the maximum flow that could be producted by existing paths if a bid was made requesting _nr and providing _np.

2.3.9 ob_fstats

```
ob_yret_stats ob_fstats()
```

gives general informations about the model:

Column	Type	Meaning
mean_time_drafts	int8	mean of delay of drafts
nb_drafts	int8	number of drafts
nb_noeuds	int8	number of bids
nb_stocks	int8	number of stocks
nb_stocks_s	int8	number of stocks type=S
nb_stocks_d	int8	number of stocks type=D
nb_stocks_a	int8	number of stocks type=A
nb_qualities	int8	number of qualities
nb_owners	int8	number of owners

all following columns should be zero

Column	Type	Meaning
unbalanced_qualities	int8	number of qualities with accounting problems
corrupted_draft	int8	number of inconsistent drafts
corrupted_stock_s	int8	number of stocks_S not related to a bid
corrupted_stock_a	int8	number of couples (quality,owner) where stocks_A is not unique

Example:

```
select * from ob_fstats()
```

2.3.10 ob_vowned

Gives quantity owned for each couple (quality,owner).

Column	Type	Meaning
qname	text	quality name
owner	text	Name of the depositary owner of this quality
qtt	int8	sum(qtt) for couples (quality,owner)
created	timestamp	min(created)
updated	timestamp	max(updated?updated:created)

examples

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

total values owned by the owner '***jack***'

2.3.11 ob_vbalance

List of values owned by user group by quality name.

Column	Type	Meaning
qname	text	quality name
qtt	int8	sum(qtt) for this quality
created	timestamp	min(created)
updated	timestamp	max(updated?updated:created)

examples:

```
SELECT * FROM ob_vbalance WHERE qname='gold';
```

Total values owned for the quality '***gold***'

```
SELECT count(*) FROM ob_vbalance WHERE qtt!=0 AND qname='gold'
```

Returns 0 if accounting is correct for this user.

2.3.12 ob_vdraft

list of commits of draft grouped by owner.

Column	Type	Meaning
did	int8	id of draft
status	char	always Draft
owner	text	owner providing the value

cntcommit	int	number of bids of the draft owned by this owner
flags	int4	bit 0 set when accepted by owner; bit 1 set when refuse by owner
created	timestamp	

usage:

```
SELECT * FROM ob_vdraft WHERE owner='paul'
```

list of drafts for the owner '***paul***' ,

```
SELECT owner,flags&1 as accepted,flags&2 as refused FROM market.vdraft WHERE did=100
```

list of partners of the draft 100 with their decisions.

2.3.13 ob_vbid

List of bids.

Column	Type	Meaning
id	int8	id of bid
owner	text	name of author of the bid, owner of the value offered
required_quality	text	
required_quantity	int8	
omega	float	the ratio provided_quantity/required quantity
provided_quality	text	Quantity of the value offered at the time the bid was created
provided_quantity	int8	
sid	int8	tstock.id f the value offered by the bid
qtt	int8	Quantity of the value offered
created	timestamp	

usage:

```
SELECT * FROM ob_vbid WHERE owner='luc'
```

list of bids of '***luc***'

2.3.14 ob_vcommit

List of commits.

Column	Type	Meaning
draft	int8	Draft id
bid	int8	Bid id
commit	int8	Commit id
owner	text	name of author , owner of the value offered

provides	text	Quality provided
qtt	int8	Quantity provided

usage:

```
SELECT * FROM ob_vcommit WHERE owner='luc'
```

list of draft commits of '***luc***'

2.3.15 ob_vmvt

returns a list of movements related to the owner.

Column	Type	Meaning
id	int8	id of the movement.
did	int8	Movements related to a given draft are referenced by a single. This is not the ob_tdraft.id, but the id of the first movement related to this draft. It is not NULL for a draft executed even if this draft where deleted. It is NULL when the movement is not created by the execution of an agreement.
provider	text	name of provider
nat	text	quality of moved value
qtt	int8	quantity moved value
receiver	text	name of receiver
created	text	timestamp

usage:

```
SELECT * from ob_vmvt where 'luc' in (provider,receiver)
```

list of movements for this owner.

2.4 Installation

2.4.1 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 installcheck
…
============== running regression test queries        ==============
test flow_1                ... ok
test flow_2                ... ok
test flow_3                ... ok
```

```
test flow_4                    ... ok

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

2.4.2 Install the model

The model is defined by the file openBarter/src/sql/model.sql. It is recommended to execute it with an admin role that is not “market”.

The model does not depend of any schema, and creates a role “market” if it doesn't exist already. All functions and views described here can be accessed only with this role.

2.5 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 postgres. 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].