The Genome Center
Washington University

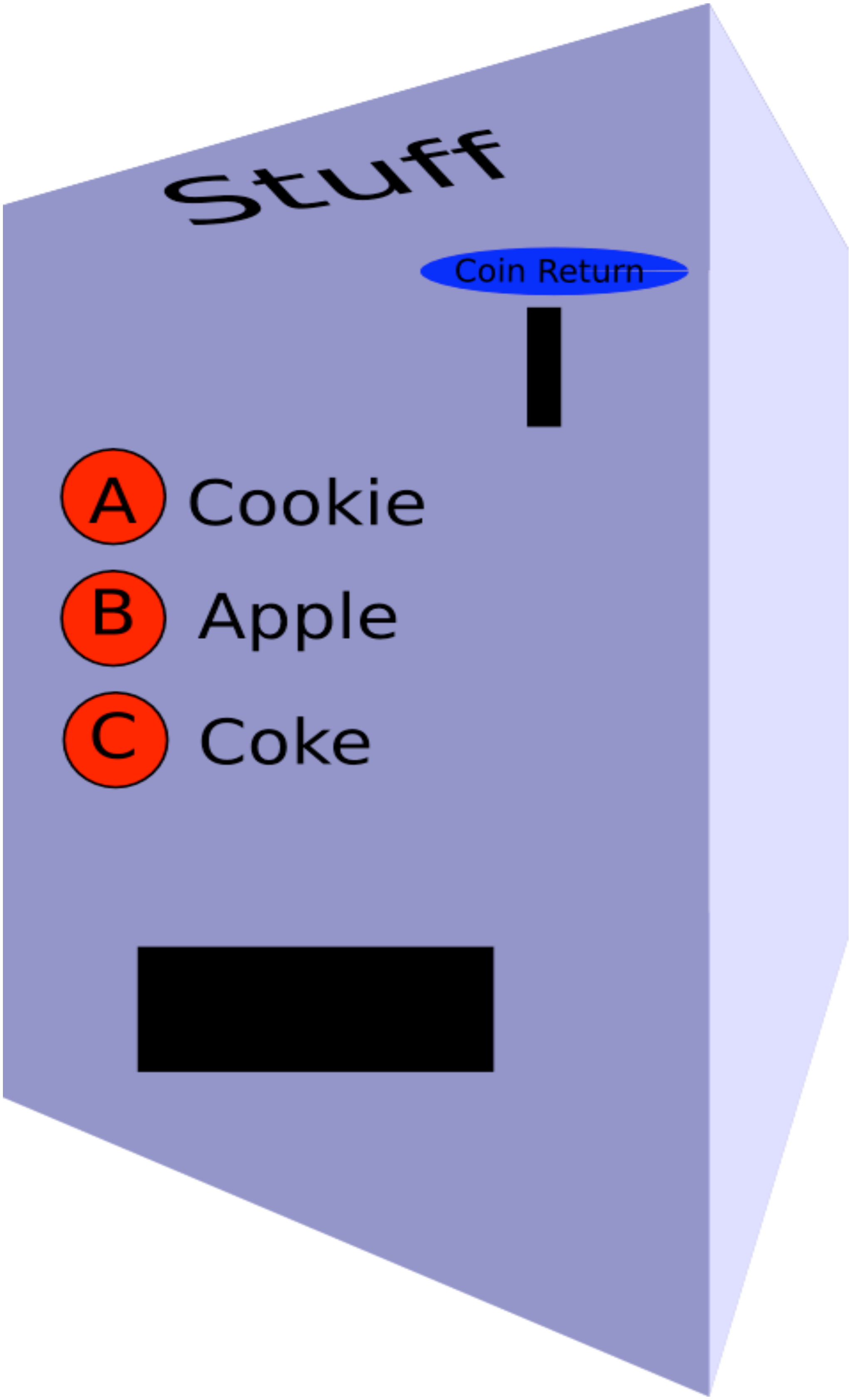
Perl/UR ORM

Anthony Brummett brummett@cpan.org

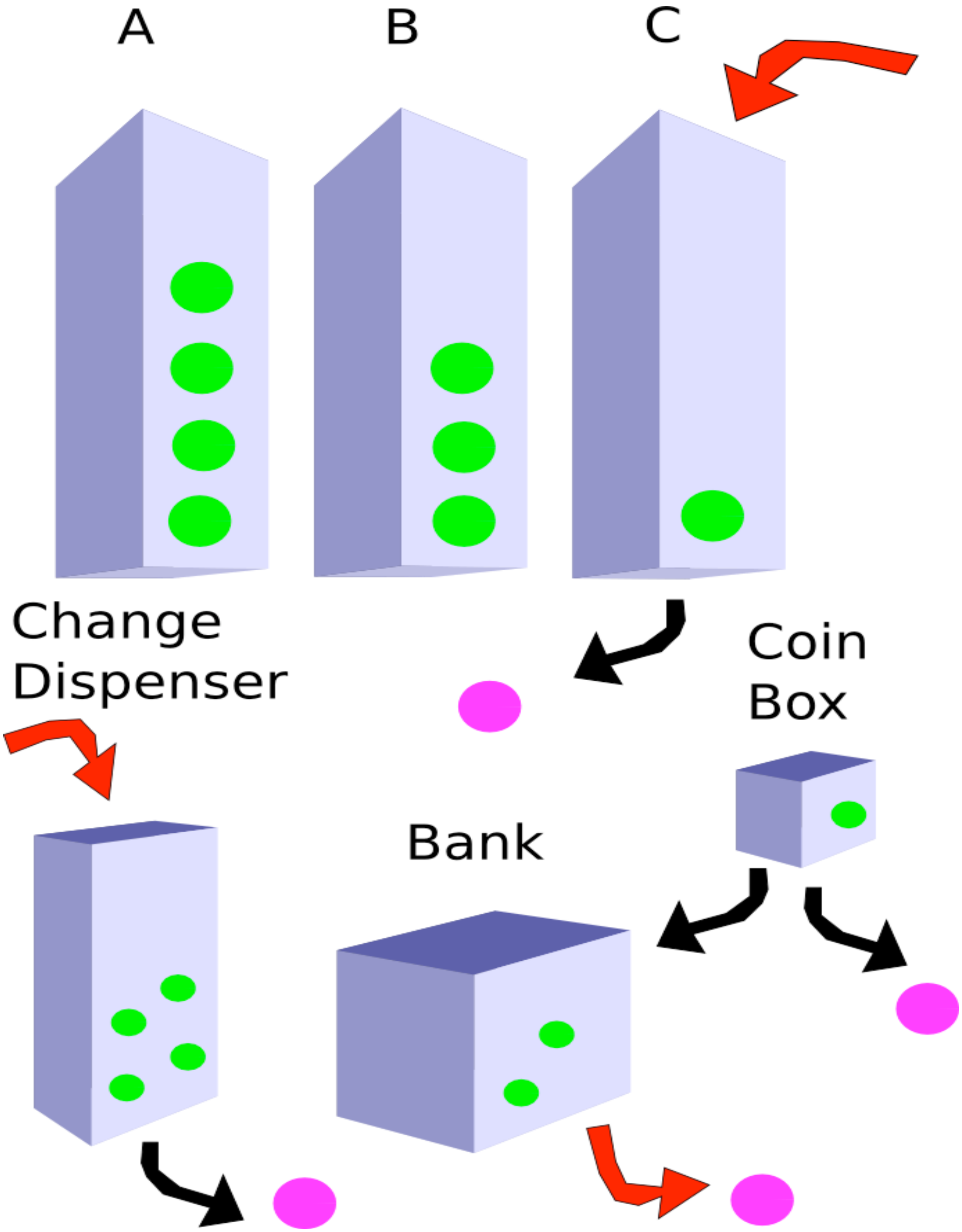
With 'use UR;' you get...

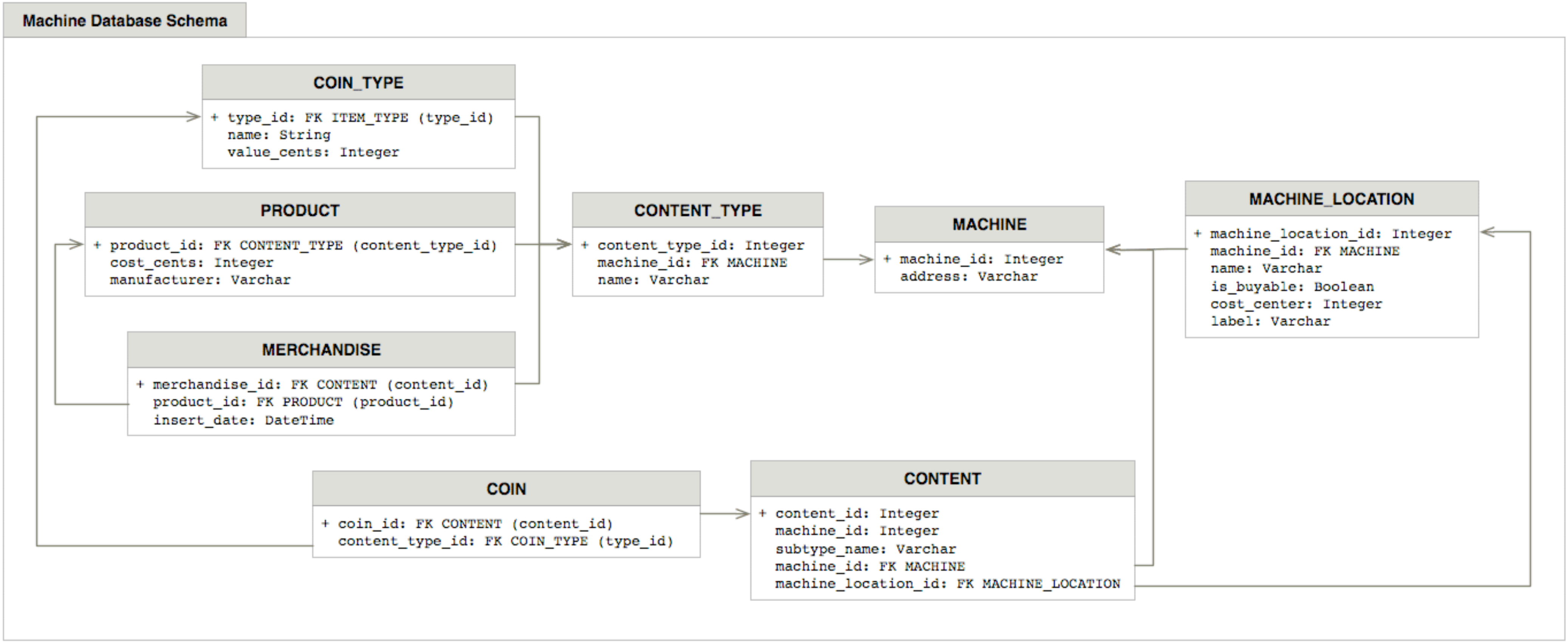
- Perl
- Command line tools to manage your metadata
- More formal class declarations
- Dynamically generated methods for class properties
- Caching between your program and its data sources
- Nestable software transactions
- Managed metadata about our schemas and synchronization with them
- Introspection: classes, properties, relationships, transactions, data sources, namespaces are objects, too.
- Infrastructure to support visualizers, aggregations of objects (Sets), command patterns

Vending machine external



Vending machine internal




Machine Database Schema
COIN_TYPE
+ type_id: FK ITEM_TYPE (type_id)
name: String
value_cents: Integer
PRODUCT
+ product_id: FK CONTENT_TYPE (content_type_id)
cost_cents: Integer
manufacturer: Varchar
MERCHANDISE
+ merchandise_id: FK CONTENT (content_id)
product_id: FK PRODUCT (product_id)
insert_date: DateTime
CONTENT_TYPE
+ content_type_id: Integer
machine_id: FK MACHINE
name: Varchar
MACHINE
+ machine_id: Integer
address: Varchar
MACHINE_LOCATION
+ machine_location_id: Integer
machine_id: FK MACHINE
name: Varchar
is_buyable: Boolean
cost_center: Integer
label: Varchar
COIN
+ coin_id: FK CONTENT (content_id)
content_type_id: FK COIN_TYPE (type_id)
CONTENT
+ content_id: Integer
machine_id: Integer
subtype_name: Varchar
machine_id: FK MACHINE
machine_location_id: FK MACHINE_LOCATION

Setting Up

```
> ur define namespace Vending
A   Vending (UR::Namespace)
A   Vending::Vocabulary (UR::Vocabulary)
A   Vending::DataSource::Meta (UR::DataSource::Meta)
A   /path/to/Vending/DataSource/Meta.sqlite3-dump (Meta DB Skeleton)
```

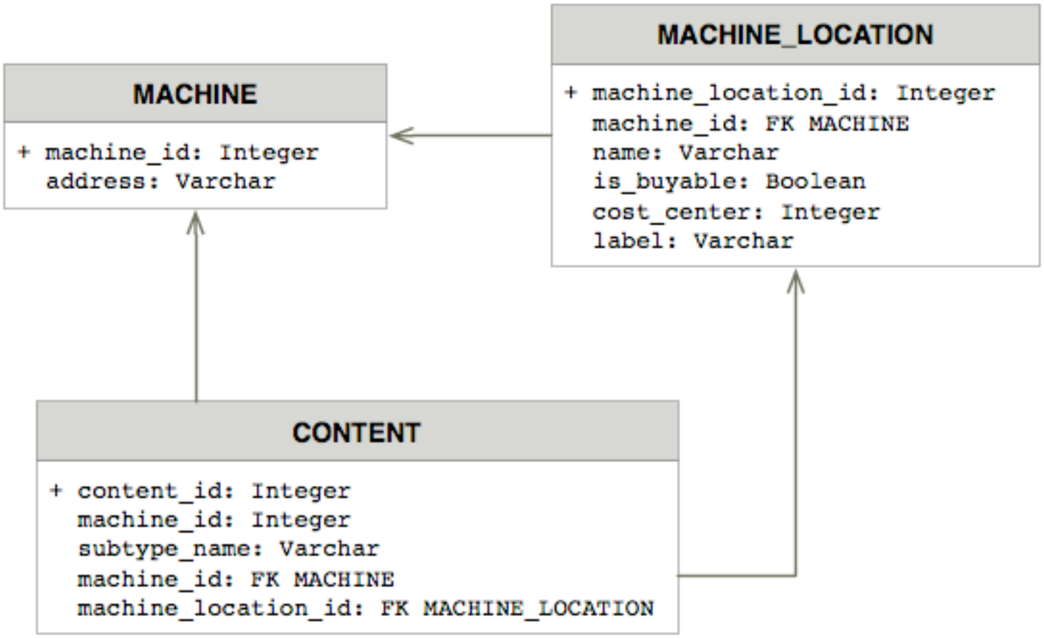
```
> ur define namespace Vending
A   Vending (UR::Namespace)
A   Vending::Vocabulary (UR::Vocabulary)
A   Vending::DataSource::Meta (UR::DataSource::Meta)
A   /path/to/Vending/DataSource/Meta.sqlite3-dump (Meta DB Skeleton)

> cd Vending
> ur define datasource sqlite —dsname Machine
A   Vending::DataSource::Machine (UR::DataSource::SQLite,UR::Singleton)
A   /path/to/Vending/DataSource/Machine.sqlite3 (empty database)
    ...connecting...
    ...ok
```

```
> ur define namespace Vending
A   Vending (UR::Namespace)
A   Vending::Vocabulary (UR::Vocabulary)
A   Vending::DataSource::Meta (UR::DataSource::Meta)
A   /path/to/Vending/DataSource/Meta.sqlite3-dump (Meta DB Skeleton)

> cd Vending
> ur define datasource sqlite —dsname Machine
A   Vending::DataSource::Machine (UR::DataSource::SQLite,UR::Singleton)
A   /path/to/Vending/DataSource/Machine.sqlite3 (empty database)
    ...connecting...
    ...ok

> sqlite3 DataSource/Machine.sqlite3
sqlite> create table MACHINE (machine_id integer NOT NULL PRIMARY KEY,
                              address varchar);
[…]
```

```
> ur define namespace Vending
A   Vending (UR::Namespace)
A   Vending::Vocabulary (UR::Vocabulary)
A   Vending::DataSource::Meta (UR::DataSource::Meta)
A   /path/to/Vending/DataSource/Meta.sqlite3-dump (Meta DB Skeleton)

> cd Vending
> ur define datasource sqlite –dsname Machine
A   Vending::DataSource::Machine (UR::DataSource::SQLite,UR::Singleton)
A   /path/to/Vending/DataSource/Machine.sqlite3 (empty database)
    ...connecting...
    ...ok

> sqlite3 DataSource/Machine.sqlite3
sqlite> create table MACHINE (machine_id integer NOT NULL PRIMARY KEY,
                              address varchar);
[…]

> ur update classes
Updating namespace: Vending
Found data sources: Machine
Checking Vending::DataSource::Machine for schema changes...
A  Machine COIN      Schema changes
A  Machine MERCHANDISE     Schema changes
A  Machine CONTENT_TYPE    Schema changes
 […]
Found 8 tables with changes.
Resolving corresponding class changes...
Updating classes...
A  Vending::Coin           uses Machine table COIN
A  Vending::Merchandise    uses Machine table MERCHANDISE
A  Vending::ContentType    uses Machine table CONTENT_TYPE
  […]
Updating class properties...
A  Vending::Coin        has new column COIN.COIN_ID (integer)
A  Vending::Coin        has new column COIN.COIN_TYPE_ID (integer)
A  Vending::Merchandise has new column MERCHANDISE.MERCHANDISE_ID (integer)
  […]
Updating class ID properties...
Updating class unique constraints...
Updating class relationships...

Saving metadata changes...
Resolved changes for 8 classes
Updating the filesystem...
A /path/to/Vending/Coin.pm
A /path/to/Vending/Merchandise.pm
A /path/to/Vending/ContentType.pm
  […]
Filesystem update complete.
Committing changes to data sources...
Cleaning up.
Update complete.

>
```

Updating classes...
results

```
> cat Content.pm

package Vending::Content;

use strict;
use warnings;

use Vending;

class Vending::Content {
    table_name => 'CONTENT',
    id_by => [
        content_id => { is => 'Integer' },
    ],
    has => [
        machine_id       => { is => 'Integer' },
        machine          => { is => 'Vending::Machine',
                              id_by => 'machine_id' },
        subtype_name     => { is => 'Text' },
        machine_location_id => { is => 'Integer' },
        machine_location => { is => 'Vending::MachineLocation,
                              id_by => 'machine_location_id' },
    ],
    schema_name => 'Machine',
    data_source => 'Vending::DataSource::Machine',
};

1;
```



Add some additional properties by hand...

```
> cat Content.pm
package Vending::Content;
use strict;
use warnings;
use Vending;

class Vending::Content {
  table_name => 'CONTENT',
  is_abstract => 1,
  subclassify_by => 'subtype_name',
  id_by => [
    content_id => { is => 'Integer' },
  ],
  has => [
    machine_id       => { is => 'Integer' },
    machine          => { is => 'Vending::Machine',
                          id_by => 'machine_id' },
    subtype_name     => { is => 'Text' },
    machine_location_id   => { is => 'Integer' },
    machine_location => { is => 'Vending::MachineLocation,
                          id_by => 'machine_location_id' },
    location_name    => { via => 'machine_location', to => 'name' },
  ],
  data_source => 'Vending::DataSource::Machine',
};
1;

> cat Coin.pm
package Vending::Coin;
use strict;
use warnings;
use Vending;

class Vending::Coin {
  table_name => 'COIN',
  is => 'Vending::Content',
  id_by => [
    coin_id => { is => 'Integer' },
  ],
  has => [
    coin_type_id     => { is => 'Integer' },
    coin_type    => { is => 'Vending::CoinType',
                     id_by => 'coin_type_id' },
    name         => { via => 'coin_type', to => 'name' },
    value_cents  => { via => 'coin_type', to => 'value_cents' },
    value_dollars => { calculate_from => 'value_cents',
                       calculate => q(
                           sprintf('$%.2f',$value_cents) )},
  ],
  data_source => 'Vending::DataSource::Machine',
};
1;
```

Change the schema

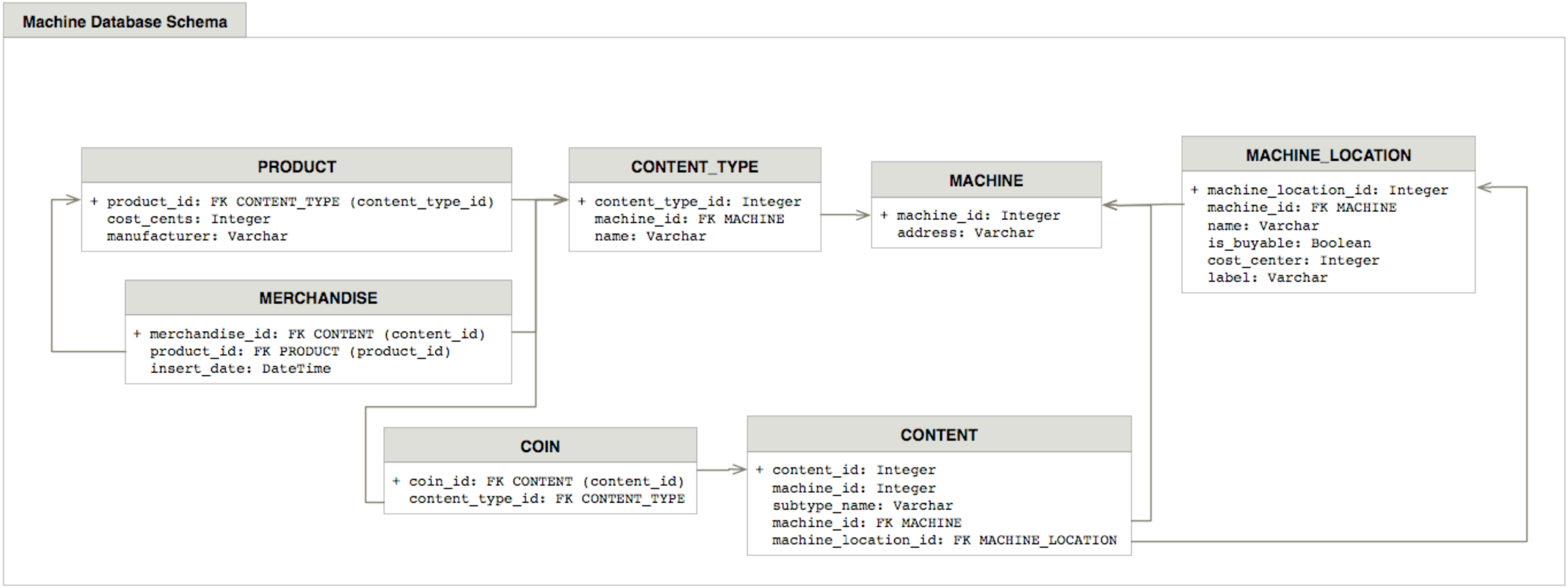
Add a new column...

```
sqlite> alter table MACHINE add column serial_number varchar;
sqlite> quit
```

Add a new column...

```
sqlite> alter table MACHINE add column serial_number varchar;
sqlite> quit
```

```
> ur update classes
Updating namespace: Vending
Found data sources: Machine
Checking Vending::DataSource::Machine for schema changes...
U  Machine MACHINE         Schema changes
Found 1 tables with changes.
Resolving corresponding class changes...
Updating classes...
Updating class properties...
A  Vending::Machine     has new column MACHINE.SERIAL_NUMBER (varchar)
Updating class ID properties...
Updating class unique constraints...
Updating class relationships...
Saving metadata changes...
Resolved changes for 1 classes
Updating the filesystem...
U /path/to/Vending/Machine.pm
Filesystem update complete.
Committing changes to data sources...
Cleaning up.
Update complete.

>
```



Machine Database Schema

PRODUCT

```
+ product_id: FK CONTENT_TYPE (content_type_id)
  cost_cents: Integer
  manufacturer: Varchar
```

CONTENT_TYPE

```
+ content_type_id: Integer
  machine_id: FK MACHINE
  name: Varchar
```

MACHINE

```
+ machine_id: Integer
  address: Varchar
```

MACHINE_LOCATION

```
+ machine_location_id: Integer
  machine_id: FK MACHINE
  name: Varchar
  is_buyable: Boolean
  cost_center: Integer
  label: Varchar
```

MERCHANDISE

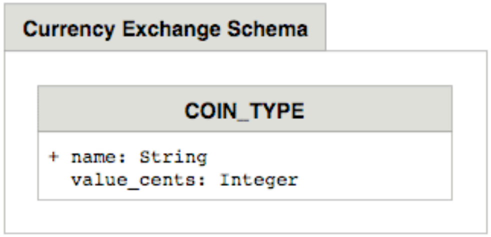
```
+ merchandise_id: FK CONTENT (content_id)
  product_id: FK PRODUCT (product_id)
  insert_date: DateTime
```

COIN

```
+ coin_id: FK CONTENT (content_id)
  content_type_id: FK CONTENT_TYPE
```

CONTENT

```
+ content_id: Integer
  machine_id: Integer
  subtype_name: Varchar
  machine_id: FK MACHINE
  machine_location_id: FK MACHINE_LOCATION
```

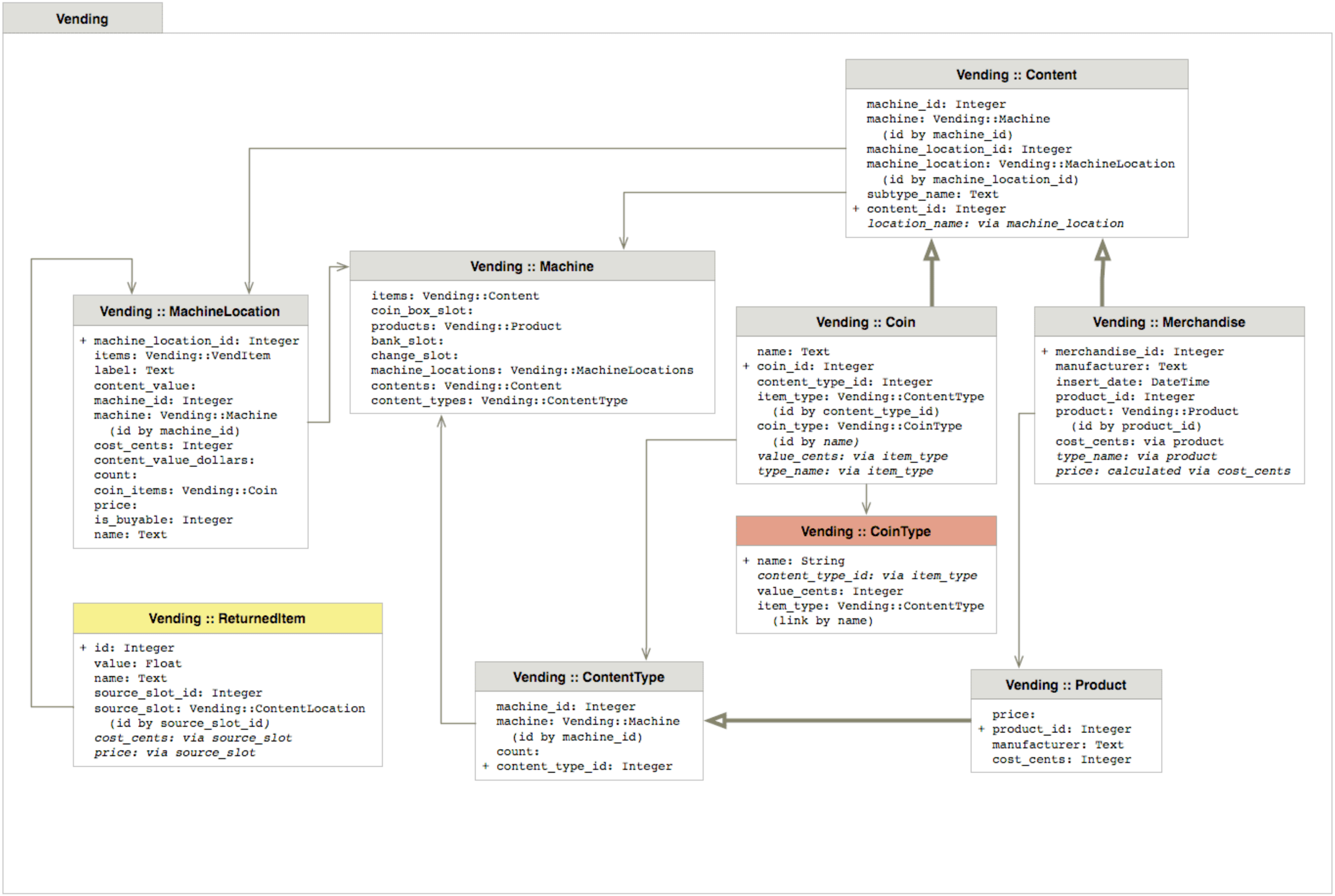


Currency Exchange Schema

COIN_TYPE

```
+ name: String
  value_cents: Integer
```

```
> cat DataSource/CoinType.pm
package Vending::DataSource::CoinType;

use strict;
use warnings;

use Vending;

my $path = '/some/nfs/path/DataSource/coin_types.tsv';

class Vending::DataSource::CoinType;
    is => ['UR::DataSource::File', 'UR::Singleton'],
    has_constant => [
        server => { value => $path },
        delimiter => { value => '\s+' },
        column_order => { value => ['name','value_cents'] },
        sort_order => { value => ['name'] },
    ],
};

1;

> cat CoinType.pm
package Vending::CoinType;

use strict;
use warnings;

use Vending;

class Vending::CoinType {
  id_by => [
      name => { is => 'Text' },
  ],
  has => [
    value_cents => { is => 'Integer' },
  ],
  data_source => 'Vending::DataSource::CoinType',
};

1;
```


Vending
Vending :: Content
machine_id: Integer
machine: Vending::Machine
(id by machine_id)
machine_location_id: Integer
machine_location: Vending::MachineLocation
(id by machine_location_id)
subtype_name: Text
+ content_id: Integer
location_name: via machine_location
Vending :: Machine
items: Vending::Content
coin_box_slot:
products: Vending::Product
bank_slot:
change_slot:
machine_locations: Vending::MachineLocations
contents: Vending::Content
content_types: Vending::ContentType
Vending :: MachineLocation
+ machine_location_id: Integer
items: Vending::VendItem
label: Text
content_value:
machine_id: Integer
machine: Vending::Machine
(id by machine_id)
cost_cents: Integer
content_value_dollars:
count:
coin_items: Vending::Coin
price:
is_buyable: Integer
name: Text
Vending :: Coin
name: Text
+ coin_id: Integer
content_type_id: Integer
item_type: Vending::ContentType
(id by content_type_id)
coin_type: Vending::CoinType
(id by name)
value_cents: via item_type
type_name: via item_type
Vending :: Merchandise
+ merchandise_id: Integer
manufacturer: Text
insert_date: DateTime
product_id: Integer
product: Vending::Product
(id by product_id)
cost_cents: via product
type_name: via product
price: calculated via cost_cents
Vending :: CoinType
+ name: String
content_type_id: via item_type
value_cents: Integer
item_type: Vending::ContentType
(link by name)
Vending :: ReturnedItem
+ id: Integer
value: Float
name: Text
source_slot_id: Integer
source_slot: Vending::ContentLocation
(id by source_slot_id)
cost_cents: via source_slot
price: via source_slot
Vending :: ContentType
machine_id: Integer
machine: Vending::Machine
(id by machine_id)
count:
+ content_type_id: Integer
Vending :: Product
price:
+ product_id: Integer
manufacturer: Text
cost_cents: Integer

Do some stuff

Easy get...

```
> export UR_DBI_MONITOR_SQL=1;

use Vending;

my $generic_type = Vending::ContentType->get(name => 'quarter');

    select CONTENT_TYPE.content_type_id, CONTENT_TYPE.name,
           CONTENT_TYPE.machine_id
    from CONTENT_TYPE
    where CONTENT_TYPE.name = ?
    order by CONTENT_TYPE.content_type_id
    params: 'quarter'
```

Get something that inherits...

```
my $cookie_type = Vending::Product->get(name => 'Cookie');

    select PRODUCT.cost_cents, PRODUCT.manufacturer,
           PRODUCT.product_id,
           CONTENT_TYPE.content_type_id, CONTENT_TYPE.name,
           CONTENT_TYPE.machine_id
    from PRODUCT
    join CONTENT_TYPE on PRODUCT.product_id =
                         CONTENT_TYPE.content_type_id
    where CONTENT_TYPE.name = ?
    order by PRODUCT.product_id
    params: 'Cookie'
```

Get something that inherits...

```
my $cookie_type = Vending::Product->get(name => 'Cookie');

    select PRODUCT.cost_cents, PRODUCT.manufacturer,
           PRODUCT.product_id,
           CONTENT_TYPE.content_type_id, CONTENT_TYPE.name,
           CONTENT_TYPE.machine_id
    from PRODUCT
    join CONTENT_TYPE on PRODUCT.product_id =
                         CONTENT_TYPE.content_type_id
    where CONTENT_TYPE.name = ?
    order by PRODUCT.product_id
    params: 'Cookie'
```

And something that inherits, by way of a delegated property...

```
my @cookies = Vending::Merchandise->get(name => 'Cookie');

  select MERCHANDISE.insert_date, MERCHANDISE.merchandise_id,
           MERCHANDISE.product_id,
           CONTENT.machine_location_id, CONTENT.subtype_name,
           CONTENT.content_id, CONTENT.machine_id
           product_1.cost_cents, product_1.manufacturer,
           product_1.product_id,
           product_2.name, product_2.content_type_id,
           product_2.machine_id
  from MERCHANDISE
  join CONTENT on MERCHANDISE.INV_ID = CONTENT.content_id
  join PRODUCT product_1 on CONTENT.product_id =
                            product_1.product_id
  join CONTENT_TYPE product_2 on product_1.product_id =
                            product_2.content_type_id
  where product_2.name = ?
  order by MERCHANDISE.merchandise_id
  params: 'Cookie'
```

Get with another kind of operator...

```
my @items = Vending::Content->get(location_name => {
                                      operator => 'like',
                                      value => 'chan%' } );

  select CONTENT.machine_location_id, CONTENT.subtype_name,
         CONTENT.content_id, CONTENT.machine_id
         machine_location_1.cost_cents, machine_location_1.is_buyable,
         machine_location_1.label, machine_location_1.name,
         machine_location_1.machine_location_id,
         machine_location_1.machine_id
  from CONTENT
  join MACHINE_LOCATION machine_location_1 on CONTENT.machine_location_id =
                                   machine_location_1.machine_location_id
  where machine_location_1.name like ?
  order by CONTENT.content_id
  params: 'chan%
```

Get with another kind of operator...

```
my @items = Vending::Content->get(location_name => {
                                      operator => 'like',
                                      value => 'chan%' } );

  select CONTENT.machine_location_id, CONTENT.subtype_name,
         CONTENT.content_id, CONTENT.machine_id
         machine_location_1.cost_cents, machine_location_1.is_buyable,
         machine_location_1.label, machine_location_1.name,
         machine_location_1.machine_location_id,
         machine_location_1.machine_id
  from CONTENT
  join MACHINE_LOCATION machine_location_1 on CONTENT.machine_location_id =
                                   machine_location_1.machine_location_id
  where machine_location_1.name like ?
  order by CONTENT.content_id
  params: 'chan%
```

** Loads a row where 'subtype_name' = 'Vending::Coin'

Get with another kind of operator...

```
my @items = Vending::Content->get(location_name => {
                                     operator => 'like',
                                     value => 'chan%' } );

  select CONTENT.machine_location_id, CONTENT.subtype_name,
         CONTENT.content_id, CONTENT.machine_id
         machine_location_1.cost_cents, machine_location_1.is_buyable,
         machine_location_1.label, machine_location_1.name,
         machine_location_1.machine_location_id,
         machine_location_1.machine_id
  from CONTENT
  join MACHINE_LOCATION machine_location_1 on CONTENT.machine_location_id =
                                  machine_location_1.machine_location_id
  where machine_location_1.name like ?
  order by CONTENT.content_id
  params: 'chan%
```

** Loads a row where 'subtype_name' = 'Vending::Coin'

Starts a parallel query to retrieve Vending::Coin objects with the same filters as the original query

```
  select COIN.coin_id, COIN.coin_type_id,
         CONTENT.machine_location_id, CONTENT.subtype_name,
         CONTENT.content_id, CONTENT.machine_id
         machine_location_1.cost_cents, machine_location_1.is_buyable,
         machine_location_1.label, machine_location_1.name,
         machine_location_1.machine_location_id,
         machine_location_1.machine_id
  from COIN
  join CONTENT on COIN.coin_id = CONTENT.content_id
  join MACHINE_LOCATION machine_location_1 on CONTENT.machine_location_id =
                                  machine_location_1.machine_location_id
  where machine_location_1.name like ?
  order by COIN.coin_id
  params: 'chan%'
```

Even more complicated...

```
my @quarters = Vending::Coin->get(value_cents => 25);
```

Even more complicated...

```
my @quarters = Vending::Coin->get(value_cents => 25);

  value_cents comes from a Vending::CoinType
    which we can query by its name
      which comes from Vending::Coin->name
        which comes from Vending::ContentType->name
      Vending::Coin is-a Vending::Content
```

Even more complicated...

```
my @quarters = Vending::Coin->get(value_cents => 25);

  value_cents comes from a Vending::CoinType
    which we can query by its name
      which comes from Vending::Coin->name
        which comes from Vending::ContentType->name
      Vending::Coin is-a Vending::Content


  select COIN.coin_id, COIN.coin_type_id,
         CONTENT.machine_location_id, CONTENT.subtype_name,
         CONTENT.content_id, CONTENT.machine_id
         coin_type_1.name, coin_type_1.content_type_id,
         coin_type_1.machine_id
  from COIN
  join CONTENT on COIN.coin_id = CONTENT.content_id
  join CONTENT_TYPE coin_type_1 on COIN.coin_type_id =
                                    coin_type_1.content_type_id
  order by COIN.coin_id
```

Even more complicated...

```
my @quarters = Vending::Coin->get(value_cents => 25);

  value_cents comes from a Vending::CoinType
    which we can query by its name
      which comes from Vending::Coin->name
        which comes from Vending::ContentType->name
      Vending::Coin is-a Vending::Content


  select COIN.coin_id, COIN.coin_type_id,
         CONTENT.machine_location_id, CONTENT.subtype_name,
         CONTENT.content_id, CONTENT.machine_id
         coin_type_1.name, coin_type_1.content_type_id,
         coin_type_1.machine_id
  from COIN
  join CONTENT on COIN.coin_id = CONTENT.content_id
  join CONTENT_TYPE coin_type_1 on COIN.coin_type_id =
                                 coin_type_1.content_type_id
  order by COIN.coin_id
```

In a parallel query, it asks the currency exchange for COIN_TYPE rows where the value is 25

```
  FILE: opened /path/to/dir/Vending/DataSource/coin_types.tsv
  FILTERS: value_cents = 25
```

Even more complicated...

```
my @quarters = Vending::Coin->get(value_cents => 25);

  value_cents comes from a Vending::CoinType
    which we can query by its name
      which comes from Vending::Coin->name
        which comes from Vending::ContentType->name
      Vending::Coin is-a Vending::Content


  select COIN.coin_id, COIN.coin_type_id,
         CONTENT.machine_location_id, CONTENT.subtype_name,
         CONTENT.content_id, CONTENT.machine_id
         coin_type_1.name, coin_type_1.content_type_id,
         coin_type_1.machine_id
  from COIN
  join CONTENT on COIN.coin_id = CONTENT.content_id
  join CONTENT_TYPE coin_type_1 on COIN.coin_type_id =
                                 coin_type_1.content_type_id
  order by COIN.coin_id
```

In a parallel query, it asks the currency exchange for COIN_TYPE rows where the value is 25

```
  FILE: opened /path/to/dir/Vending/DataSource/coin_types.tsv
  FILTERS: value_cents = 25
```

And internally does the equivalent of an SQL join between the first and second queries' rows on CONTENT_TYPE.name = COIN_TYPE.name

Returns Vending::Coin objects back to the caller.

Caching and Context

Lazily talking to the database:

```
# Loads info from the database
my @products = Vending::Product->get();
```

Lazily talking to the database:

```
# Loads info from the database
my @products = Vending::Product->get();

# This is a subset of something already loaded
# won't ask the database
my $cookie = Vending::Product->get(name => 'Cookie');
```

Lazily talking to the database:

```
# Loads info from the database
my @products = Vending::Product->get();

# This is a subset of something already loaded
# won't ask the database
my $cookie = Vending::Product->get(name => 'Cookie');


$cookie->price_cents(55);
```

Lazily talking to the database:

```
# Loads info from the database
my @products = Vending::Product->get();

# This is a subset of something already loaded
# won't ask the database
my $cookie = Vending::Product->get(name => 'Cookie');


$cookie->price_cents(55);

# Won't update the database yet

my $coke = Vending::Product->get(name => 'Coke')
$coke->delete();
```

Lazily talking to the database:

```
# Loads info from the database
my @products = Vending::Product->get();

# This is a subset of something already loaded
# won't ask the database
my $cookie = Vending::Product->get(name => 'Cookie');


$cookie->price_cents(55);

# Won't update the database yet

my $coke = Vending::Product->get(name => 'Coke')
$coke->delete();

# Still not updating...
```

Lazily talking to the database:

```
# Loads info from the database
my @products = Vending::Product->get();

# This is a subset of something already loaded
# won't ask the database
my $cookie = Vending::Product->get(name => 'Cookie');


$cookie->price_cents(55);

# Won't update the database yet

my $coke = Vending::Product->get(name => 'Coke')
$coke->delete();

# Still not updating...

if (UR::Context->commit()) {
    # Now it's back at the database
    return 1;
}
```

Lazily talking to the database:

```
# Loads info from the database
my @products = Vending::Product->get();

# This is a subset of something already loaded
# won't ask the database
my $cookie = Vending::Product->get(name => 'Cookie');


$cookie->price_cents(55);

# Won't update the database yet

my $coke = Vending::Product->get(name => 'Coke')
$coke->delete();

# Still not updating...

if (UR::Context->commit()) {
    # Now it's back at the database
    return 1;
} else {
    # Coke Merchandise still remains
    # There were constraint problems...
    my @changed_objects = grep { $_->changed }
                          UR::Object->all_objects_loaded();
    # Fix them up?
}
```

Lazily talking to the database:

```
# Loads info from the database
my @products = Vending::Product->get();

# This is a subset of something already loaded
# won't ask the database
my $cookie = Vending::Product->get(name => 'Cookie');


$cookie->price_cents(55);

# Won't update the database yet

my $coke = Vending::Product->get(name => 'Coke')
$coke->delete();

# Still not updating...

if (UR::Context->commit()) {
    # Now it's back at the database
    return 1;
} else {
    # Coke Merchandise still remains
    # There were constraint problems...
    my @problem_objects = grep { $_->invalid }
                          grep { $_->changed }
                          UR::Object->all_objects_loaded();
    # Fix them up?
}
```

Lazily talking to the database:

```
# Loads info from the database
my @products = Vending::Product->get();

# This is a subset of something already loaded
# won't ask the database
my $cookie = Vending::Product->get(name => 'Cookie');


$cookie->price_cents(55);

# Won't update the database yet

my $coke = Vending::Product->get(name => 'Coke')
$coke->delete();

# Still not updating...

if (UR::Context->commit()) {
    # Now it's back at the database
    return 1;
} else {
    # Coke Merchandise still remains
    # There were constraint problems...
    # Just go back to the way we were...
    UR::Context->rollback();
}
```

Properties and their Accessors

Meanwhile... in the Vending::Machine...

```
  has => [
    products => { is => 'Vending::Product',
                  reverse_id_by => 'machine',
                  is_many => 1 },
    machine_locations => { is => 'Vending::MachineLocation',
                  reverse_id_by => 'machine',
                  is_many => 1 },
    change   => { via => 'machine_locations',
                  to => '-filter',
                  where => [name => 'change'] },
    coin_box => { via => 'machine_locations',
                  to => '-filter',
                  where => [name => 'box'] },
  ],
```

Meanwhile... in the Vending::Machine...

```
  has => [
    products => { is => 'Vending::Product',
                  reverse_id_by => 'machine',
                  is_many => 1 },
    machine_locations => { is => 'Vending::MachineLocation',
                  reverse_id_by => 'machine',
                  is_many => 1 },
    change   => { via => 'machine_locations',
                  to => '-filter',
                  where => [name => 'change'] },
    coin_box => { via => 'machine_locations',
                  to => '-filter',
                  where => [name => 'box'] },
  ],


my @products = $machine->products();

my $cookie = $machine->products(name => 'Cookie');

my $iter = $machine->product_iterator(cost_cents => {
                                          operator => '<',
                                          value => '110' });
while (my $obj = $iter->next()) {
    # Do something
}

my $product = $machine->add_product(manufacturer => 'Acme',
                                    cost_cents => 1234,
                                    name => 'dynamite');
```

Vending Machine at Work

```
# Put in some money

sub insert {
    my($self,$item_name) = @_;

    my $coin_type = Vending::CoinType->get(name => $item_name);
    unless ($coin_type) {
        $self->error_message(“This machine does not accept '$item_name'”);
        return;
    }
    my $coin_box = $self->coin_box();
    my $coin = $coin_box->add_coin_item(type_id => $coin_type->type_id,
                                        machine_id => $self->machine_id);
    return defined($coin);
}
```

```
# Put in some money

sub insert {
    my($self,$item_name) = @_;

    my $coin_type = Vending::CoinType->get(name => $item_name);
    unless ($coin_type) {
        $self->error_message(“This machine does not accept '$item_name'”);
        return;
    }
    my $coin_box = $self->coin_box();
    my $coin = $coin_box->add_coin_item(type_id => $coin_type->type_id,
                                        machine_id => $self->machine_id);
    return defined($coin);
}

# Get our coins back from the Vending::Machine...

sub coin_return {
    my $self = shift;

    my $coin_box = $self->coin_box;
    my @coins = $coin_box->items;
    my @returned_coins = Vending::ReturnedItems->create_from_items(@coins);

    $_->delete foreach @coins;
    return @returned_coins;
}
```

Buy something from Vending::Machine...

```
sub buy {
    my($self,@location_names) = @_;

    my $inserted_money = $self->coin_box->content_value();

    my @bought_items ;

    my $transaction = UR::Context::Transaction->begin();

    eval {
        foreach my $loc_name ( @location_names ) {
            my $vend_ = $self->machine_locations(name => $loc_name);
            my $item_iter = $vend_->item_iterator();

            my $item = $item_iter->next();
            if (!$item) {
                die “We're out of $_name”;
            }

            $inserted_money -= $item->cost_cents;
            if ($inserted_money < 0) {
                die “You did not insert enough money”;
            }

            my $bought = Vending::ReturnedItem->create_from_item($item);
            $item->delete();

            push @bought_items, $bought;
        };
        # make_change will die if there's not enough change
        push @bought_items, $self->make_change($inserted_money);
    };

    if ($@) {
        # There was an exception
        $transaction->rollback();
        $self->error_message(“Couldn't process your purchase: $@”);
        return;
    } else {
        # Everything worked
        $transaction->commit();
        $self->status_message(“OK”);
        return @bought_items;
    }
}

# In another part of the code

$machine->insert('dollar');
my @received = $self->buy('a','b');
print “You get “,scalar(@received), “things:\n”;
foreach my $thing ( @received ) {
    print $thing->name,”\n”;
}


UR::Context->commit();
```

Command Pattern

Reusable work units:

```
> cat Command/Outputter.pm
package Vending::Command::Outputter;
use strict;
use warnings;
use Vending;

class Vending::Command::Outputter {
    is_abstract => 1,
    is => 'Vending::Command',
    doc => 'abstract parent for things that output items to the user'
};

sub execute {
    my $self = shift;
    my @user_items = $self->_get_items_to_output();

    foreach my $item ( @user_items ) {
        print “You get: “,$item->name,”\n”;
    }
    return 1;
}

1;
```

Reusable work units:

```
> cat Command/Outputter.pm
package Vending::Command::Outputter;
use strict;
use warnings;
use Vending;

class Vending::Command::Outputter {
    is_abstract => 1,
    is => 'Vending::Command',
    doc => 'abstract parent for things that output items to the user'
};

sub execute {
    my $self = shift;
    my @user_items = $self->_get_items_to_output();

    foreach my $item ( @user_items ) {
        print “You get: “,$item->name,”\n”;
    }
    return 1;
}

1;

> cat Command/CoinReturn.pm
package Vending::Command::CoinReturn;
use strict;
use warnings;
use Vending;

class Vending::Command::CoinReturn {
   is => 'Vending::Command::Outputter',
   doc => 'Return all inserted coins back to the user',
};

sub _get_items_to_output {
    my $self = shift;

    my $machine = $self->machine;
    my @items = $machine->coin_return();
    return @items;
}

1;
```

Instant command-line interface:

```
> cat vend
#!/usr/bin/perl

use strict;
use warnings;

use Vending;

Vending::Command->execute_with_shell_params_and_exit();

>
```

Instant command-line interface:

```
> cat vend
#!/usr/bin/perl

use strict;
use warnings;

use Vending;

Vending::Command->execute_with_shell_params_and_exit();

> ./vend
Commands for Vending
  buy          Attempt to get a sellable item
  coin-return  Return all inserted coins back to the customer
  dime         Insert a dime into the machine
  dollar       Insert a dollar into the machine
  insert-money Insert a non-standard coin type
  menu         Show the items available to buy
  nickel       Insert a nickel into the machine
  quarter      Insert a quarter into the machine
  service      Service-mode commands
>
```

Instant command-line interface:

```
> cat vend
#!/usr/bin/perl

use strict;
use warnings;

use Vending;

Vending::Command->execute_with_shell_params_and_exit();

> ./vend
Commands for Vending
  buy          Attempt to get a sellable item
  coin-return  Return all inserted coins back to the customer
  dime         Insert a dime into the machine
  dollar       Insert a dollar into the machine
  insert-money Insert a non-standard coin type
  menu         Show the items available to buy
  nickel       Insert a nickel into the machine
  quarter      Insert a quarter into the machine
  service      Service-mode commands
> ./vend dollar
> ./vend menu
NAME     LABEL     PRICE
----     -----     -----
a        Cookie    $0.65
b        Apple     $1.00
c        Coke      $1.50
You have inserted $1.00 so far
>
```

Instant command-line interface:

```
> cat vend
#!/usr/bin/perl

use strict;
use warnings;

use Vending;

Vending::Command->execute_with_shell_params_and_exit();

> ./vend
Commands for Vending
  buy          Attempt to get a sellable item
  coin-return  Return all inserted coins back to the customer
  dime         Insert a dime into the machine
  dollar       Insert a dollar into the machine
  insert-money Insert a non-standard coin type
  menu         Show the items available to buy
  nickel       Insert a nickel into the machine
  quarter      Insert a quarter into the machine
  service      Service-mode commands
> ./vend dollar
> ./vend menu
NAME     LABEL     PRICE
----     -----     -----
a        Cookie    $0.65
b        Apple     $1.00
c        Coke      $1.50
You have inserted $1.00 so far
> ./vend coin-return
You get: dollar
>
```

Use them in a program, too:

```
use Vending;

$dollar = Vending::Command::Dollar->create();
$dollar->execute();

$coin_return = Vending::Command::coin_return->create();
$coin_return->execute();

UR::Context->commit();
```

Ompaloskepsis

(or... Contemplating one's own navel)

(or... Introspection)

(or... Metadata)

Classes are ~~people~~ objects too...

```
my $product_meta = UR::Object::Type->get(class_name => 'Vending::Product);
```

Classes are ~~people~~ objects too...

```
my $product_meta = UR::Object::Type->get(class_name => 'Vending::Product);

# UR::Object::Property objects named id, property_id, content_type_id,
# name, manufacturer, cost_cents, count, price, etc.
my @all_product_properties = $product_meta->all_property_metas();

# UR::Object::Property objects named property_id, manufacturer,
# cost_cents and price
my @new_product_properties = $product_meta->direct_property_metas();

# Only the properties we inherit from all parent classes
my @inherited_properties = $product_meta->ancestry_property_metas();

# our direct parent classes
my @immediate_parent_classes = $product_meta->parent_class_metas();

# All parent classes, their parents, their parents, etc...
my @all_inherited_parent_classes = $product_meta->ancenstry_class_metas();
```

Classes are ~~people~~ objects too...

```
my $product_meta = UR::Object::Type->get(class_name => 'Vending::Product);

# UR::Object::Property objects named id, property_id, content_type_id,
# name, manufacturer, cost_cents, count, price, etc.
my @all_product_properties = $product_meta->all_property_metas();

# UR::Object::Property objects named property_id, manufacturer,
# cost_cents and price
my @new_product_properties = $product_meta->direct_property_metas();

# Only the properties we inherit from all parent classes
my @inherited_properties = $product_meta->ancestry_property_metas();

# our direct parent classes
my @immediate_parent_classes = $product_meta->parent_class_metas();

# All parent classes, their parents, their parents, etc...
my @all_inherited_parent_classes = $product_meta->ancenstry_class_metas();



# A particular property meta-object
my $property_meta = $product_meta->property_meta_for_name('name');

# The class it's attached to
print $property_meta->class_name(), “\n”;
my $class_meta = $property_meta->class_meta();

print $property_meta->data_type(), ”\n”;
print $property_meta->property_name(), “\n”;
# also via(), to, where, is_id, is_delegated, is_optional, is_calculated...
```

Classes are ~~people~~ objects too...

```
my $product_meta = UR::Object::Type->get(class_name => 'Vending::Product);

# UR::Object::Property objects named id, property_id, content_type_id,
# name, manufacturer, cost_cents, count, price, etc.
my @all_product_properties = $product_meta->all_property_metas();

# UR::Object::Property objects named property_id, manufacturer,
# cost_cents and price
my @new_product_properties = $product_meta->direct_property_metas();

# Only the properties we inherit from all parent classes
my @inherited_properties = $product_meta->ancestry_property_metas();

# our direct parent classes
my @immediate_parent_classes = $product_meta->parent_class_metas();

# All parent classes, their parents, their parents, etc...
my @all_inherited_parent_classes = $product_meta->ancestry_class_metas();



# A particular property meta-object
my $property_meta = $product_meta->property_meta_for_name('name');

# The class it's attached to
print $property_meta->class_name(), “\n”;
my $class_meta = $property_meta->class_meta();

print $property_meta->data_type(), ”\n”;
print $property_meta->property_name(), “\n”;
# also via(), to, where, is_id, is_delegated, is_optional, is_calculated...


# For delegated/object properties, UR::Object::Reference
my @relationships = $class_meta->reference_metas;

# Unique constraints, UR::Object::Property::Unique
my @unique_metas = $class_meta->direct_unique_metas();
```

Or get them directly...

```
my @properties = UR::Object::Property->get(property_name => 'type_id');

my $property = UR::Object::Property->get(class_name => 'Vending::Coin',
                                         property_name => 'type_id');

my @int_properties = UR::Object::Property->get(data_type => 'Integer');


my @bar_classes = UR::Object::Type->get(table_name => 'BAR');

my @baz_properties = UR::Object::Property->get(column_name => 'BAZ');
```

Or get them directly...

```
my @properties = UR::Object::Property->get(property_name => 'type_id');

my $property = UR::Object::Property->get(class_name => 'Vending::Coin',
                                         property_name => 'type_id');

my @int_properties = UR::Object::Property->get(data_type => 'Integer');


my @bar_classes = UR::Object::Type->get(table_name => 'BAR');

my @baz_properties = UR::Object::Property->get(column_name => 'BAZ');
```

UR is introspectional, too...

```
my $ur_namespace = UR::Namespace->get('UR');

my $property_meta_class = UR::Object::Type->get(
                               class_name => 'UR::Object::Property
                         );

my $property_meta_property = UR::Object::Property->get(
                                class_name => 'UR::Object::Type',
                                property_name => 'ancestry_property_metas',
                              );
```

Where do I come from...

```
my $class_meta = Vending::Machine->get_class_object();

my $table = $class_meta->direct_table_meta();
   $table = UR::DataSource::RDBMS::Table->get(
                   table_name => $class_meta->table_name
            );

my @tables = $class_meta->all_table_metas();


my $property_meta = $class_meta->property_meta_for_name('machine_id');
my $column = $property_meta->table_column_meta;
   $column = UR::DataSource::RDBMS::TableColumn->get(
                   table_name => $class_meta->table_name,
                   column_name => $property_meta->column_name
             );
```

Novel classes on the fly...

```
use Vending;
UR::Object::Type->define(
    class_name => 'New::Thing',
    id_by => [
        thing_id => { is => 'Integer' },
    ],
    has => [
        product => { is => 'Vending::Product', id_by => 'product_id' },
        name => { via => 'product', to => 'name' },
    ],
);

# product_id 2 has name = 'Apple'
New::Thing->create(product_id => 2);

my $same_new_thing = New::Thing->get(name => 'Apple');
```

Thanks.

UR was built by the software development team at the Washington University Genome Center. Incarnations of it run laboratory automation and analysis systems for high-throughput genomics.

Scott Smith
Todd Hepler
Craig Pohl
Todd Hepler
Ben Oberkfell
Kevin Crouse
Adam Dukes
Indraniel Das
Shin Leong
Eddie Belter
Ken Swanson
Scott Abbott
Alice Diec
William Schroeder
Eric Clark
Shawn Leonard
Lynn Carmichael
Jason Walker
Amy Hawkins
Gabe Sanderson
James Weible
James Eldred
Michael Kiwala
Mark Johnson
Kyung Kim
Jon Schindler
Justin Lolofie
Chris Harris
Jerome Peirick
Ryan Richt
John Osborne
David Dooling