Data Migration with Python

Richard.House@i-logue.com

10 May 2011

Aim

- To share an approach developed in Python to migrate data from numerous sources into a brand new system database.

Outline

- Background
- Choice of tools
- Code that proved useful

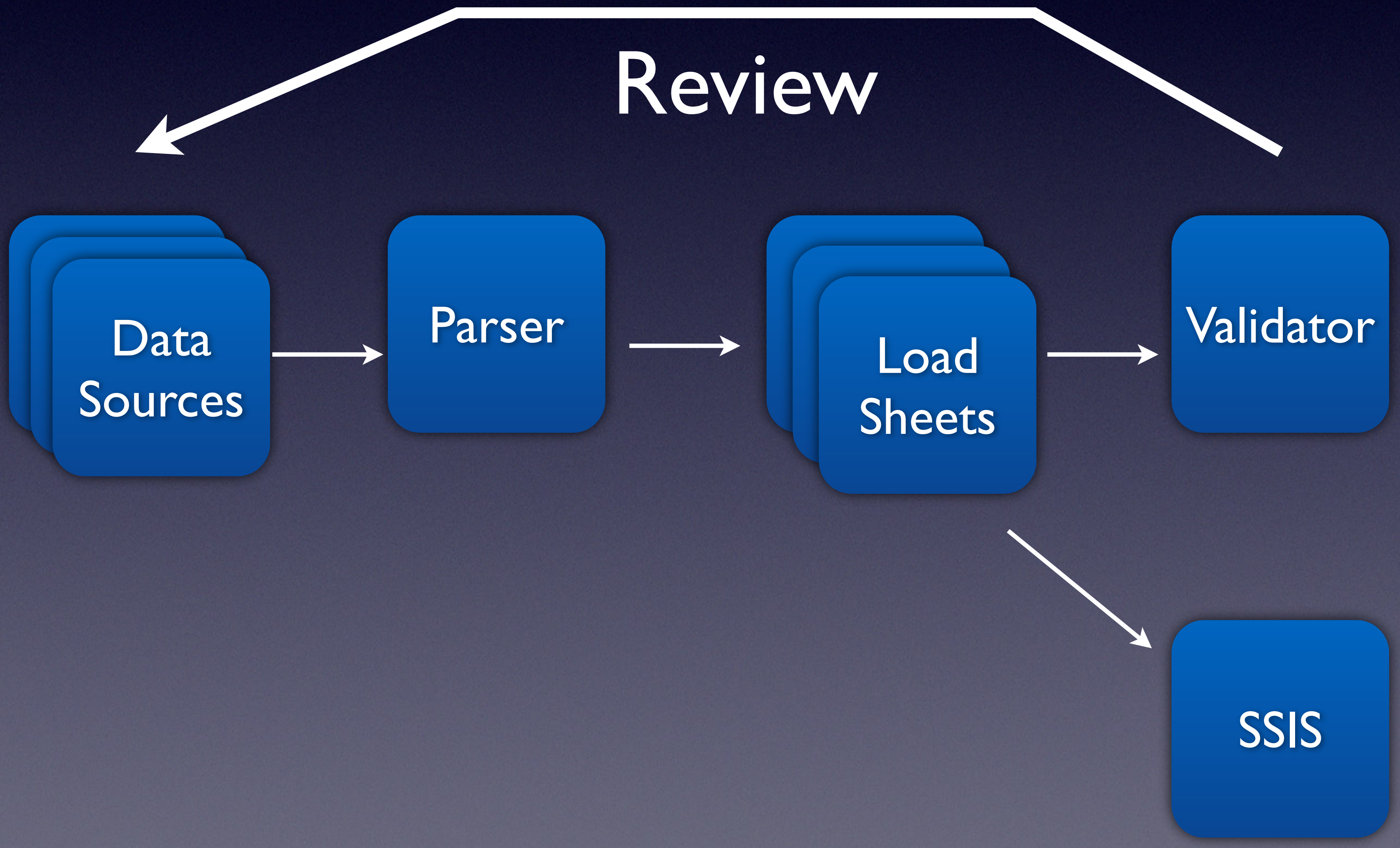
Background

- Target system: MS SSIS interface
- Numerous data sources (via xls, csv)
- Issues: data quality, semantic changes
- Timescale: 6 weeks to go-live

Features

- Hands free conversions
- Eyeball free data verification
- Review many, write once
- Concise error summaries
- Focus on mappings rather than source data changes
- Hands free operation - scripted
- Django database temporary, regularly rebuilt

Architecture



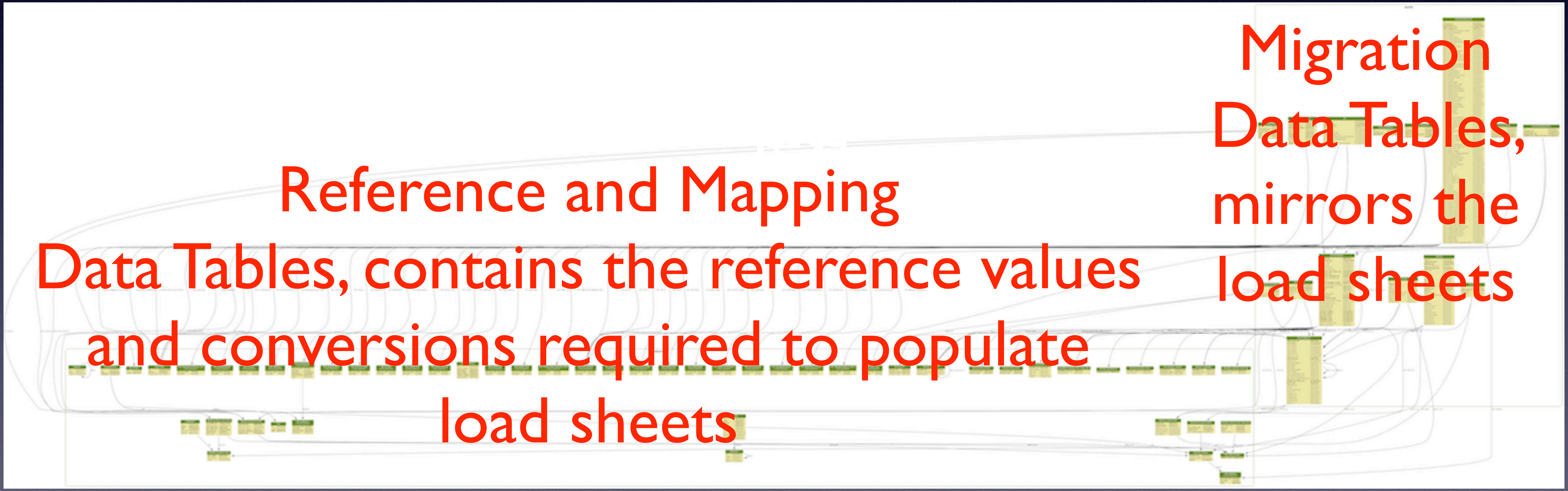
Tools

- xlrd
- xlwt
- database Django or SQLAlchemy?
- Google Map API -address cleaning

Why Django

- I didn’t have time to learn something new
- So much “Out-of-the-box” enables a quick start

Schema - Scope


Reference and Mapping Data Tables, contains the reference values and conversions required to populate load sheets
Migration Data Tables, mirrors the load sheets

Useful Django Features

- Model validators:
  - from django.core import validators
- Setting/checking dependent values
  - override clean()

Useful Python Features

- dicts for:
  - handling row data
  - default and mapping values

Handling Excel Data I

```
def _parse_row(self, sheet, row_index, date_as_tuple=False, strip_space=True):
    """ Sanitize incoming excel data """
    # Data Type Codes:
    EMPTY = 0
    TEXT = 1 # a Unicode string
    NUMBER = 2 #float
    DATE = 3 #float
    BOOLEAN = 4 #int; 1 means TRUE, 0 means FALSE
    ERROR = 5
    values = []
    # Get cell format indexes.
    row_xfs = [sheet.cell_xf_index(row_index, col)
            for col in xrange(0, sheet.ncols)]
    try:
        for type, value, xf in zip(
                sheet.row_types(row_index),
                sheet.row_values(row_index),
                row_xfs):
            if type == NUMBER:
                # Convert to integer if a round number.
                if value == int(value):
                    value = int(value)
                # Check for percentage
                # Fixme: Percentage format key not correctly matching 9 or 10?.
                cell_format_key = self.book.xf_list[xf].format_key
                # Check if format string contains a %, e.g. 0.00%.
                if self.book.format_map[cell_format_key].format_str.find(r'%') > -1:
                    # Convert from fraction to percentage number
                    value *= 100.0
                    # import pdb; pdb.set_trace()
```

Handling Excel Data2

```
            elif type == DATE:
                # If date ambiguous,treat as number.
                datetuple = xlrd.xldate_as_tuple(value, self.book.datemode)
                if date_as_tuple:
                    value = datetuple
                else:
                    # time only no date component
                    if datetuple[0] == 0 and datetuple[1] == 0 and \
                       datetuple[2] == 0:
                        value = "%02d:%02d:%02d" % datetuple[3:]
                    # date only, no time
                    elif datetuple[3] == 0 and datetuple[4] == 0 and \
                         datetuple[5] == 0:
                        value = "%04d-%02d-%02d" % datetuple[:3]
                    else: # full date
                        value = "%04d-%02d-%02d %02d:%02d:%02d" % datetuple
            elif type == TEXT and strip_space == True:
                value = value.strip()
            elif type == ERROR:
                value = xlrd.error_text_from_code[value]
            values.append(value)
        return values
    except xlrd.xldate.XLDateAmbiguous:
        print type, value, xf
        print "Check for dates 01/01/1900."
        raise
```

Cleaning Addresses

```
def google_address(address_string, country=None):
    """Queries the Google Maps API geodecoder with address text.
    country is a two letter country code, e.g. gb.
    Parses the returned json file for address fields.
    http://code.google.com/apis/kml/articles/geocodingforkml.html
    http://code.google.com/apis/maps/documentation/geocoding/
    >>> gdata=google_address('250 ST GEORGES TERRACE, Perth')
    >>> print gdata
    {'city': u'Perth', 'address1': u'250', 'address2': u'St Georges Terrace', 'state':
u'WA', 'postal_code': u'6000', 'country': u'AU'}

    ...
```

Free service, daily cap, with rate limit. $ if you need more.

Handling Foreign Key Relationships

- Reference data set up in related tables/ models. Values populated from a reference data spreadsheet.
- Related object found through:
  - Direct Foreign Keys
  - Uniques
  - Unique togethers

Django Field Utilities

```python
def get_unique_fields(model):
    """Returns list of unique fields on model that are not the primary key."""
    return [f.name for f in model._meta.fields if (f.unique == True)]

def get_mandatory_fields(model):
    """Returns a list of field names that cannot be null."""
    # blank attribute used rather than null, as char fields default to '', not null.
    return [f.name for f in model._meta.fields if (f.blank == False)]
```

Populating Foreign Keys

```python
def populate_foreignkeys(model, d, ignore_fields=None, default_values=None, match_fields=None):
    """Replace foreign key values with the objects they reference for
    the specified model. Tries to match on any unique keys in the
    related model, not just the primary key.
    match_fields ((field, sort_order)) are tuples indicating any further
    fields to search. If more than one is found the first one [0] is
    matched based on the sort order.
    Returns modified keys if no errors, otherwise raises a ValidationError
    with a dictionary of errors.
    """
    if ignore_fields is None:
        ignore_fields = []
    if default_values is None:
        default_values = {}
    if match_fields is None:
        match_fields = ()
    errors = {}
    mandatory_fields = get_mandatory_fields(model)
    for field in model._meta.fields:
        try:
            found_flag = False # True when referenced object found.
            # Delete any empty string fields to force validation checks later.
            if d[field.name] is None or d[field.name] == '': # an empty value.
                d[field.name] = default_values.get(field.name, '') # set any default.
                if d[field.name] == '':
                    del d[field.name] # Remove blank fields.
                    continue # next field.
```

Populating Foreign Keys2

```
# If field is a foreign key, get the related objects.
if isinstance(field, models.ForeignKey):
    unique_togethers = field.rel.to._meta.unique_together # Tupple of tupples of unique_together fields
    unique_fields = get_unique_fields(field.rel.to)
    # Search all unique fields in related model.
    if unique_fields:
        # Get values in row for unique together fields and search related model.
        for k in unique_fields:
            try:
                params = {k: d[field.name]}
                cache_key = ':'.join(['fk',model.__name__, k, field.name, str(d[field.name])])
                if cache_key not in cache:
                    # Get an object on the related model.
                    rm = field.rel.to.objects.get(**params)
                    cache.set(cache_key, rm)
                # Replace dictionary value with object.
                d[field.name] = cache.get(cache_key)
                found_flag = True
                break # Match found, so break out of loop
            except ObjectDoesNotExist:
                continue
            except ValueError: # e.g. invalid literal for int() with base 10: 'No'
                # import pdb; pdb.set_trace() # Debug
                continue
```

Populating Foreign Keys3

```
        if unique_togethers and not found_flag:
            for ut in unique_togethers:
                try:
                    # Construct search parameters.
                    params1 = dict([(f, d[f]) for f in ut])
                    cache_key = ':'.join(['fk', model.__name__, '_'.join(ut), field.name, '_'.join([('%s'%d[f])
for f in ut]).encode('ascii','backslashreplace')])
                    if cache_key not in cache:
                        # Get an object on the related model.
                        rm = field.rel.to.objects.get(**params1)
                        cache.set(cache_key, rm)
                    d[field.name] = cache.get(cache_key)
                    break # Match found, so break out of loop
                except ObjectDoesNotExist:
                    continue
            else: # no match found in for loop search, so raise exception if blank are not allowed.
                # Try match_fields defaults.
                for rel_model, j, sort_field in match_fields:
                    if rel_model != field.rel.to:
                        continue
                    try:
                        params = {'%s__iexact'%j: d[field.name]}
                        rm = field.rel.to.objects.filter(**params).order_by(sort_field)[0]
                        d[field.name] = rm
                        break # Match found.
                    except IndexError: # list index [0] out of range, nothing found.
                        continue
                else:
                    # import pdb; pdb.set_trace() # Debug
                    if field.null == False:
                        msg = '<%s> is not a recognised value. Check mappings.'%(field.name)
                        errors.setdefault('%s:%s'%(model.__name__, field.name), []).append(msg)
                    else:
                        d[field.name] = None
```

Populating Foreign Keys3

```
            # Check that the field value is an object of the correct type.
            if not isinstance(d[field.name], field.rel.to):
                # import pdb; pdb.set_trace() # Debug
                msg = '<%s> object with value not found.'%(field.name)
                errors.setdefault('%s:%s'%(model.__name__, field.name), []).append(msg)
    except KeyError, err: # Field is not in sheet column header.
        if field.name in ignore_fields: # OK to ignore.
            pass
        if field.name in mandatory_fields:
            # import pdb; pdb.set_trace() # Debug
            msg = 'No value found for mandatory field.'
            errors.setdefault('%s:%s'%(model.__name__, field.name), []).append(msg)
        else:
            pass
    except: # Other problems
        raise
if errors:
    raise ValidationError(errors)
return d
```

Error Reporting

```python
class ErrorLog(object):
    """Logs individual errors, and sumarises byt type.
    {row_no: {'field_name': error_message, ...},...}

    >>> el = ErrorLog()
    >>> el.add(1, {'name':['Too long'],'description': ['Too short']})
    >>> el.add(2, {'name':['Too long','All upper case'], 'date': ['Invalid']})
    >>> el.add(3, {'name':['Too long'], 'date': ['In the future']})
    >>> print el.array()
    [['Field', 'Error', 'Number', 'Rows'], ['date', 'In the future', 1, '3'], ['date', 'Invalid', 1, '2'],
['description', 'Too short', 1, '1'], ['name', 'All upper case', 1, '2'], ['name', 'Too long', 3, '1,2,3']]
    """
    def __init__(self):
        self.d = defaultdict(dict)
        self.inv = dict()

    def add(self, row_no, error_dict):
        self.d[row_no].update(error_dict)

    def number(self):
        """Returns number of errors added."""
        return len(self.d)

    def invert(self):
        """Invert the log
        {'field_name': {'error_message': [row_no,...]}...}
        """
        for row, errors in self.d.iteritems():
            for f, err_list in errors.iteritems():
                for err in err_list:
                    f_dict = self.inv.setdefault(f, {})
                    f_dict.setdefault(err,[])
                    self.inv[f][err].append(row)

    def array(self):
        "Return array of errors."
        self.invert()
        array = []
        for f, err_dict in self.inv.iteritems():
            for err, rows in err_dict.iteritems():
                array.append([f, err, len(rows), abbr_seq(rows)])
        array.sort()
        array.insert(0,['Field','Error','Qty', 'Row Numbers'])
        return array
```

Summary

- Django has useful data migration functionality.
- xlrd & xlwt make it easy to work with business users.