ETL Migration

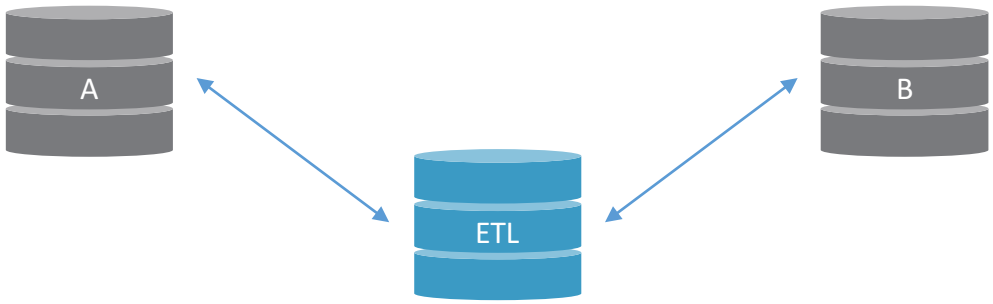
ODOO VERSION 7.0 MIGRATION ANALYSIS DOCUMENTATION

Table of Content

Introduction to ETL

ETL is an Odoo module developed to move data between databases easily. It can be used for data migration from different Odoo versions (v7 to v8, or v7 to v9), data synchronization between Odoo databases, joining of Odoo databases, and also cleaning data from Odoo databases.

ETL is an abbreviation of extract, transform, and load running on the intermediate database such that the following schema apply:



In which database A can be the source Odoo database version X and database B can be target/destination Odoo database version Y.

There are advantages of using the ETL module such as following:

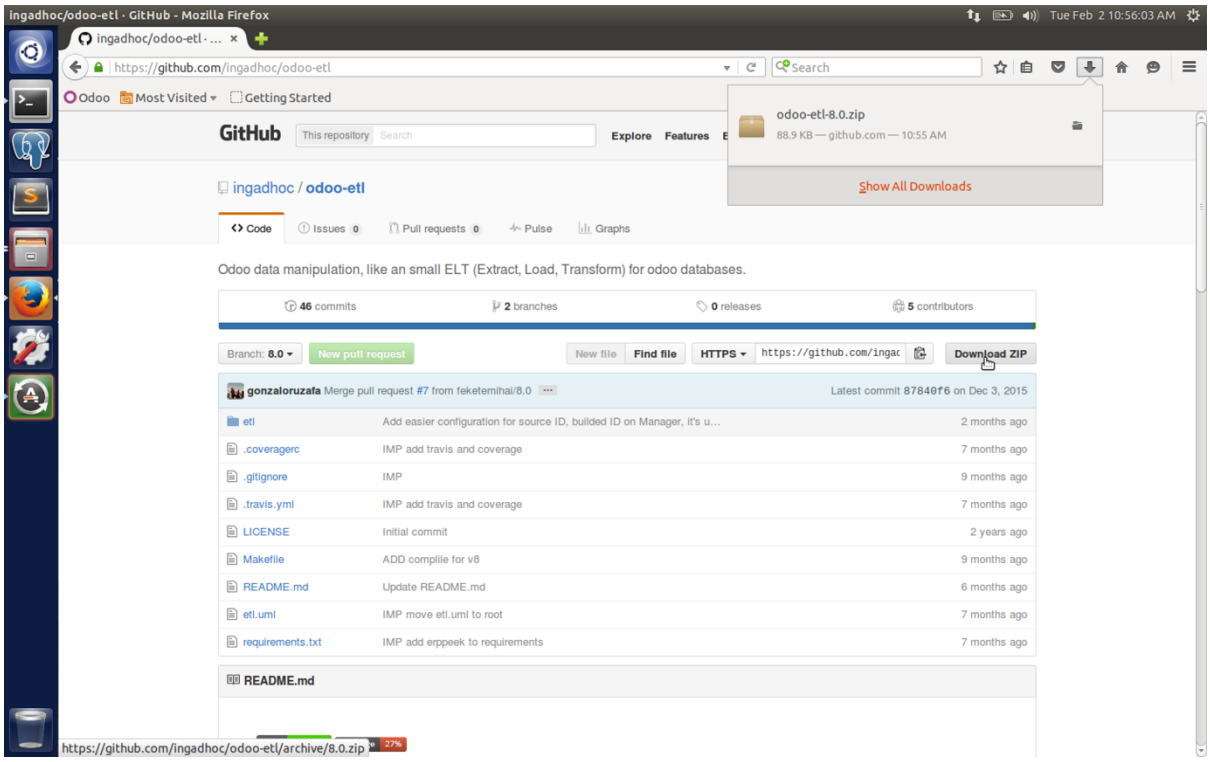
- Can be used by functional consultants
- Simple development, native Odoo methods
- Odoo module
- Works in most Odoo major version (v6, v7, v8)
    - Also works fine in v9 requiring a little modification to the python code
- Multiple uses as mentioned earlier

More details will be available at *https://www.dropbox.com/s/wvj2pse0irenel2/ETL%20-%20%20Move%20data%20between%20Odoo%20databases%20easily.pdf?dl=0*.

Resolving Error when Setting Odoo Version 9.0 as Target Database

The error is caused by the change of the transient field name at Odoo v9 from *osv_memory* into *transient*. The ETL module use the domain *[('osv_memory', '=', False)]* to filter out transient models when performing the *Read Model* action. It's therefore necessary to modify the code at *manager.py* python script located inside the ETL addons folder to allow the migration to Odoo version 9.0 as shown in step 2 of migration.

Step 1: Download the ETL module Source Code



Download the ETL module of branch 8.0 from Adhoc Group's ETL GitHub at *https://github.com/ingadhoc/odoo-etl*.

After making sure that Odoo server is not running, copy and paste the *etl* module folder from the downloaded folder into the addons folder of Odoo v9.

Step 2: Modify the *manager.py*

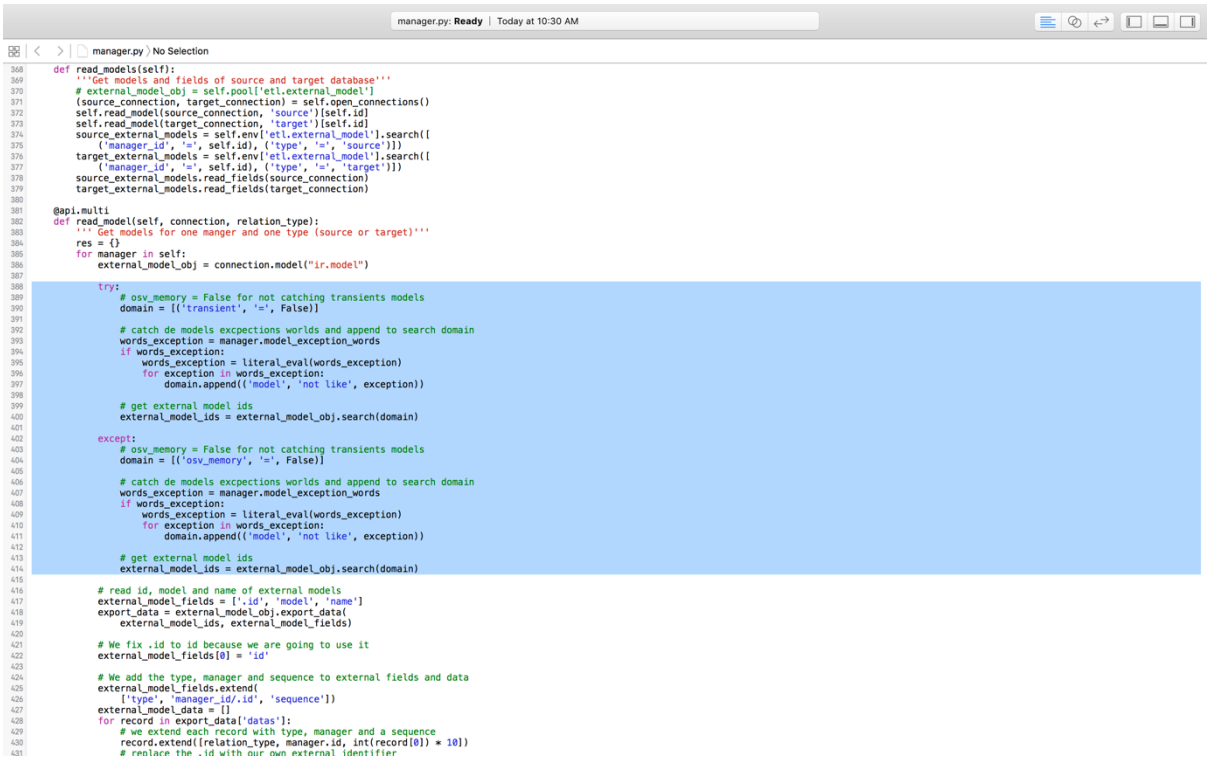
Modify the *manager.py* python script at line **388** located at the ETL addons folder to allow setting version 9.0 Odoo as the target/destination database.

Replace:

```
            # osv_memory = False for not catching transients models
            domain = [('osv_memory', '=', False)]

            # catch de models excpections worlds and append to search
domain
            words_exception = manager.model_exception_words
            if words_exception:
                words_exception = literal_eval(words_exception)
                for exception in words_exception:
                    domain.append(('model', 'not like', exception))

            # get external model ids
            external_model_ids = external_model_obj.search(domain)
```


Into:

```
            try:
                # osv_memory = False for not catching transients models
                domain = [('transient', '=', False)]

                # catch de models excpections worlds and append to search
domain
                words_exception = manager.model_exception_words
                if words_exception:
                    words_exception = literal_eval(words_exception)
                    for exception in words_exception:
                        domain.append(('model', 'not like', exception))

                # get external model ids
                external_model_ids = external_model_obj.search(domain)

            except:
                # osv_memory = False for not catching transients models
                domain = [('osv_memory', '=', False)]

                # catch de models excpections worlds and append to search
domain
                words_exception = manager.model_exception_words
                if words_exception:
                    words_exception = literal_eval(words_exception)
                    for exception in words_exception:
                        domain.append(('model', 'not like', exception))

                # get external model ids
                external_model_ids = external_model_obj.search(domain)
```



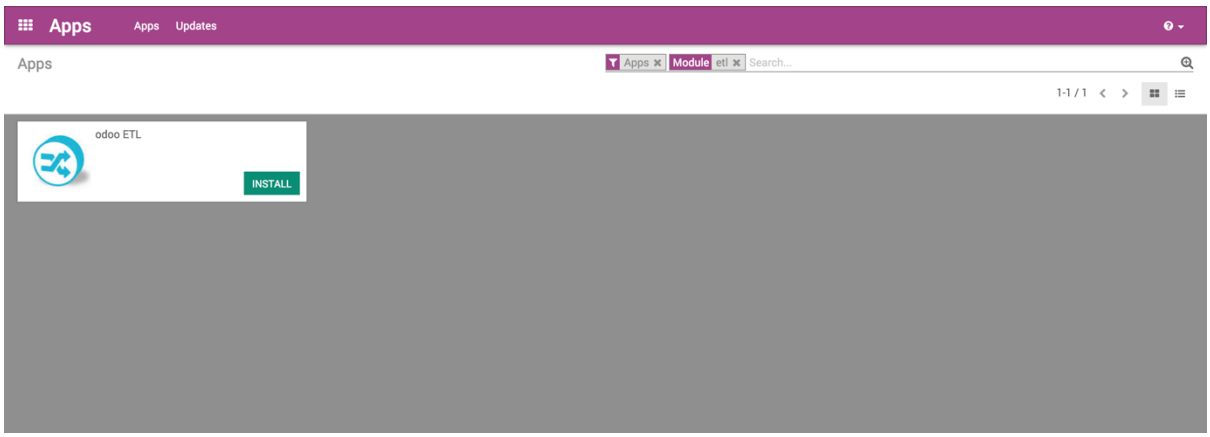
Step 3: Install ETL Module Dependency

Simply run the following command line at terminal to install and confirm the dependency:

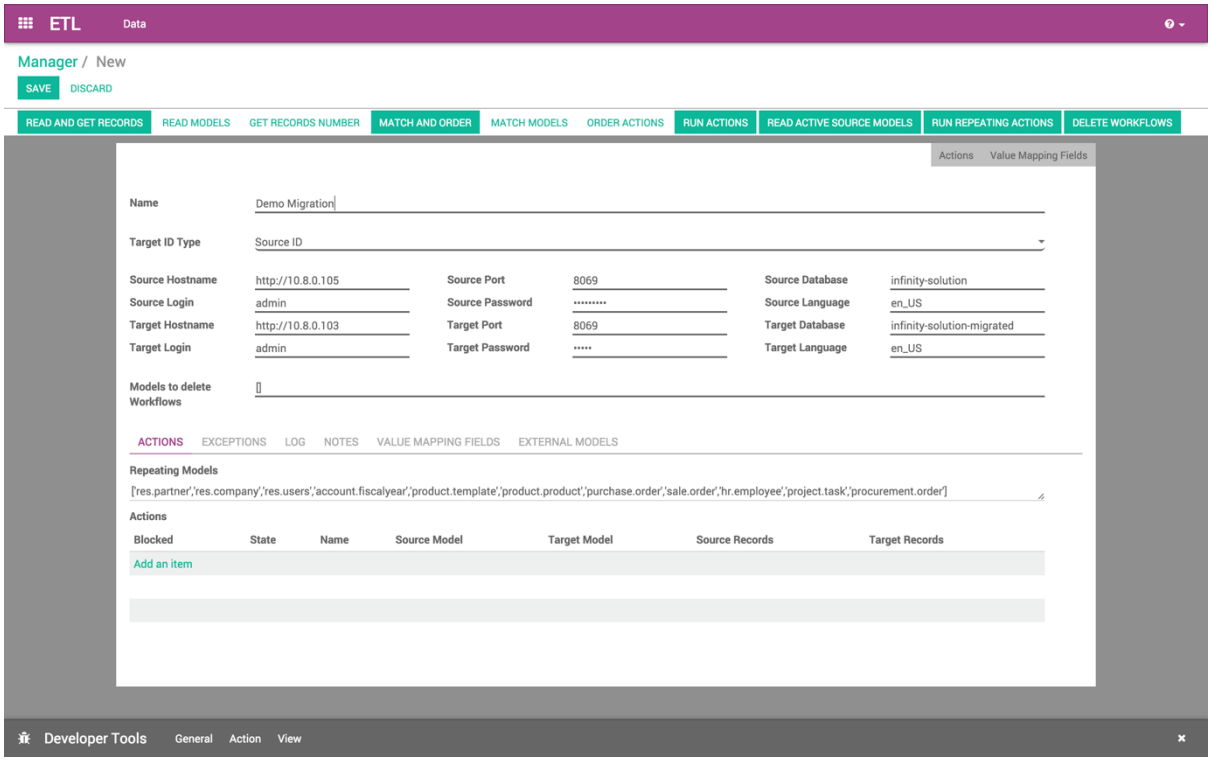
```
sudo apt-get update
sudo apt-get install python-pip
sudo pip install erppeek
```

Step 4: Setup the ETL Database

Start the Odoo server then create a new empty database without the demo data and install the ETL Module



Step 6: Configure ETL Manager



After installing ETL module, navigate to ETL's manager model and create a new manager with the following details:

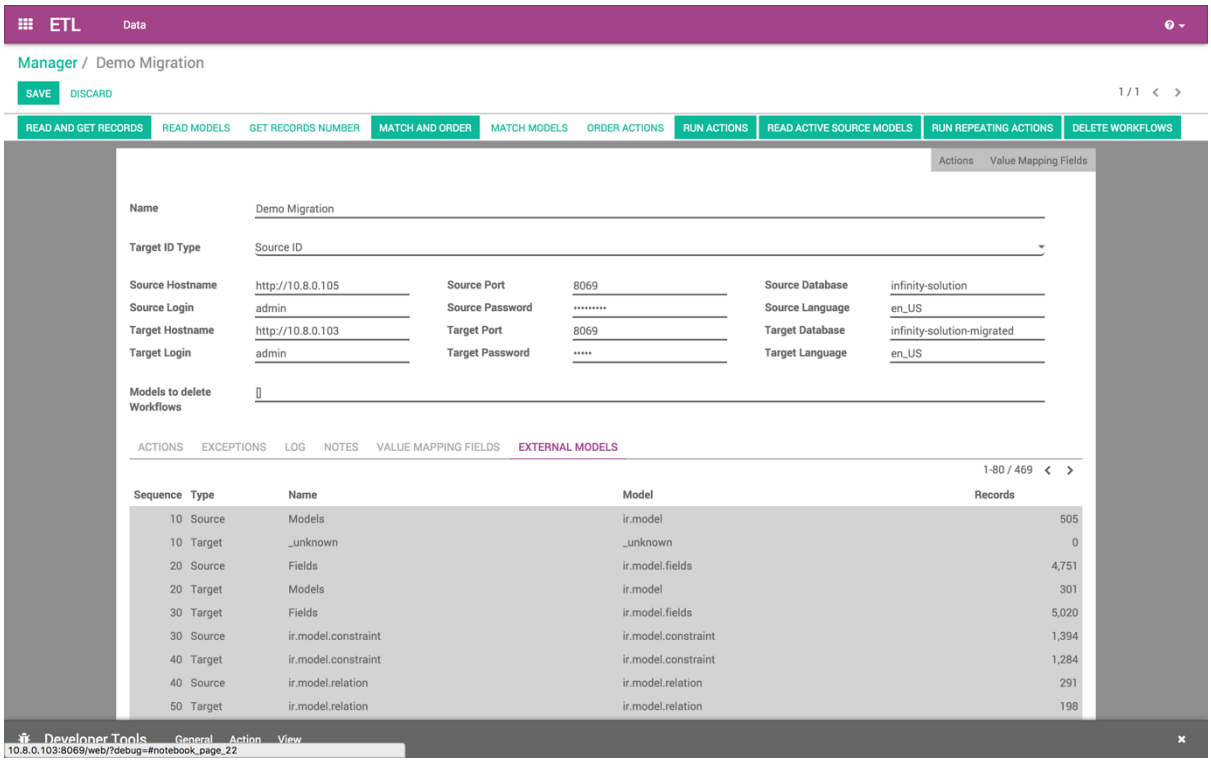
- *Name* field contains the name of your manager without any specific restriction for the name.
- *Target ID Type* field is a selection field with possible values Source ID or Builded ID. When set to Source ID, the record XML ID that will be used in the migration process will be according to the default source exported external ID. On the other hand, when set to Builded ID, the record XML ID that will be used in the migration process will be customized according to the prefix set later at an additional field. The usage of source ID is recommended when performing migration process.
- *Source Hostname* field should be the source database host URL that is used to access the Odoo database from remote OS. For example: *http://192.168.1.101*.
- *Source Port* field should be the source database port that is used to access the Odoo database. For example: *8069*.
- *Source Database* field should be the source database name.
- *Source Login* field should be the username that is used to login to the source database from the login page. Make sure the user have a full access to all the models.
- *Source Password* field should be the password according to the username that is used to login to the source database from the login page.
- *Source Language* field is the source database default language. It's recommended to keep the language as default (en_US).
- *Target Hostname* field should be the target database host URL that is used to access the Odoo database from remote OS. For example: *http://192.168.1.101*.
- *Target Port* field should be the target database port that is used to access the Odoo database. For example: *8069*.
- *Target Database* field should be the target database name.
- *Target Login* field should be the username that is used to login to the target database from the login page. Make sure the user have a full access to all the models.
- *Target Password* field should be the password according to the username that is used to login to the target database from the login page.
- *Target Language* field is the target database default language. It's recommended to keep the language as default (en_US).

Step 7: Preparing the Target Database for Migration

Make sure the target database is created and all the modules that will receive the records from the source database is installed. For example if you're performing the migration for HR and Projects, make sure the HR and Projects module is installed in the target/destination database and of course the source database.

Step 8: Read Databases

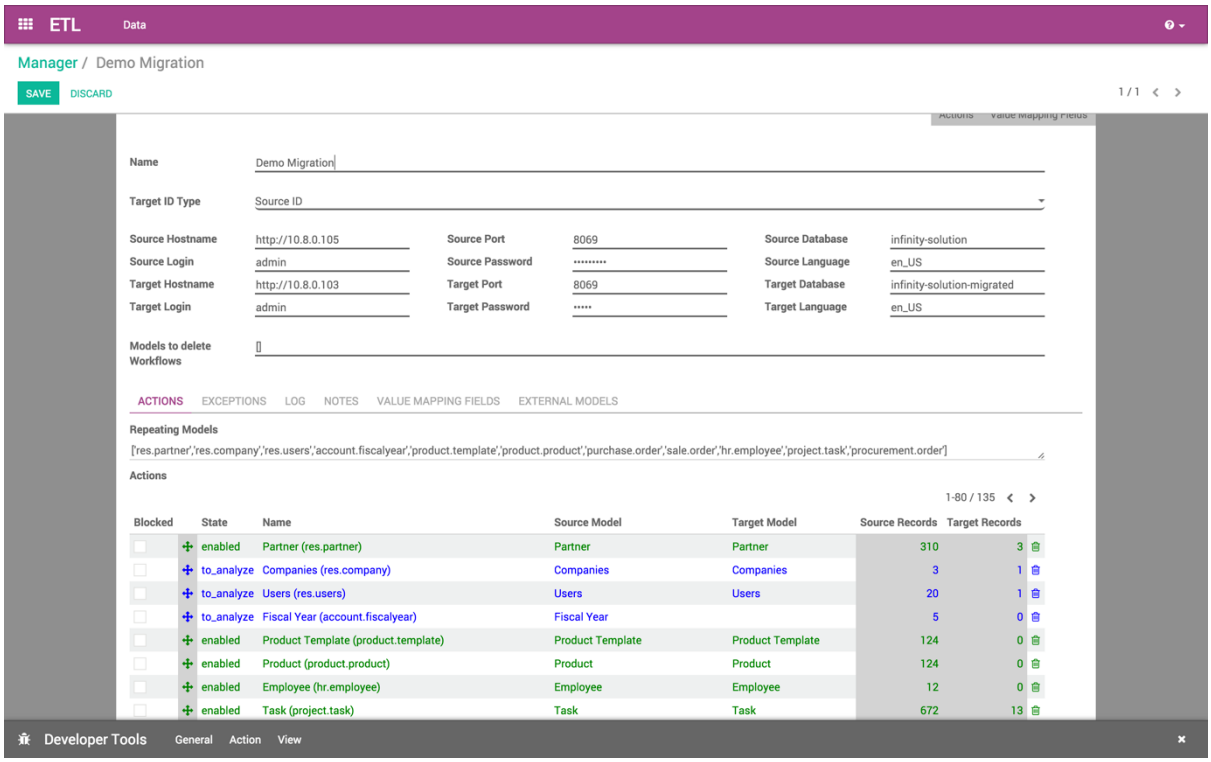
To read the models and get the record counts from the source and the target/destination database click *Read and Get Record* from the action bar. The ETL module will now attempt to connect and read from the source and destination database.



After the process is done, the *External Models* tab from your manager form view should contain the list of models that have been read from the source and target database (along with its fields when clicked) and record counts.

Step 9: Mapping

Matching the source models and the target models along with its fields can be done automatically by ETL; however, the result may not be perfectly correct. Some models and fields that changes across the version may have to be manually matched which will be explained in the next section. To perform an automatic model and fields mapping, simply click *Match and Order* from the action bar.

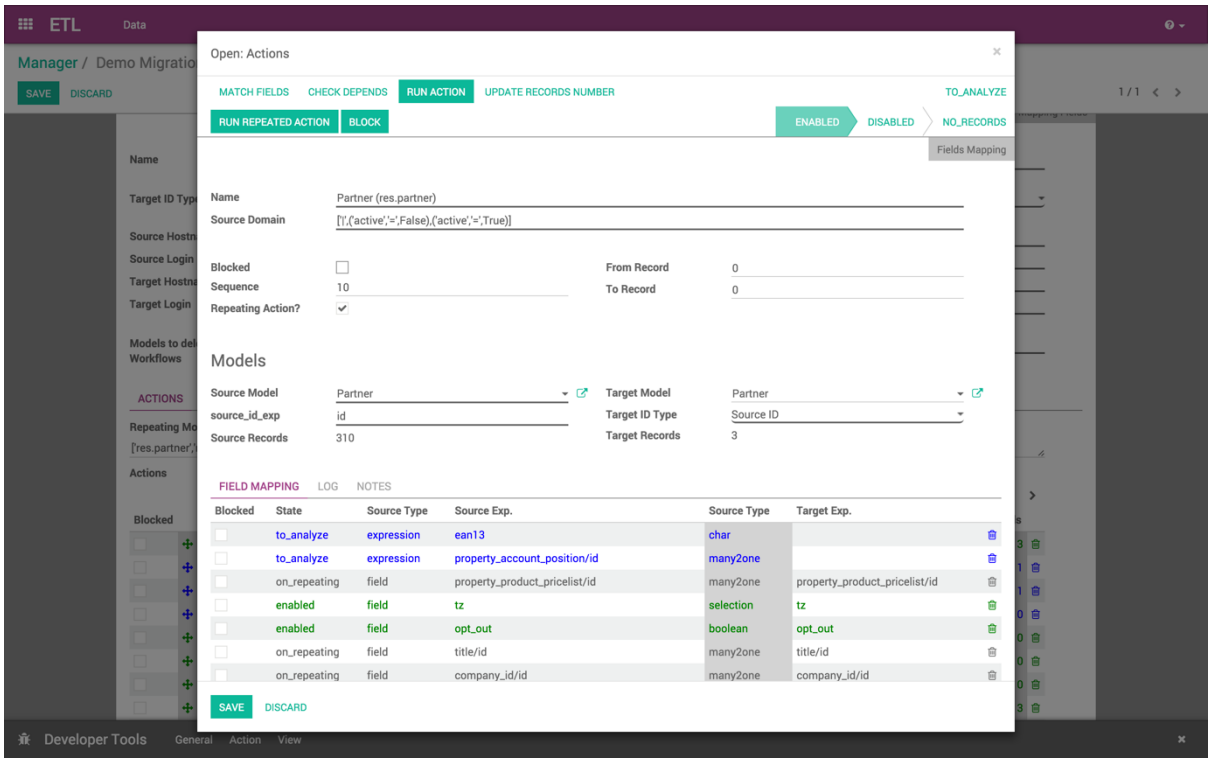


After the process is done, the *Actions* tab from your manager form view should contain the list of actions (model mappings) that have been matched and ordered by ETL.

Step 10: Test Actions

At the first use of the ETL manager, it's necessary to test the actions one by one which also means the migration will happen model by model for the first time. An action represents a migration for a single model at a time. Actions can also be understood as model mapping. It's not necessary to configure all the actions/model mapping implied by the *Match and Order* action, but only the required actions/model mapping necessary for the intended migration.

To be able to configure the actions and test it, simply click it from the list of actions in the manager.



Following is the details about the fields in the action model:

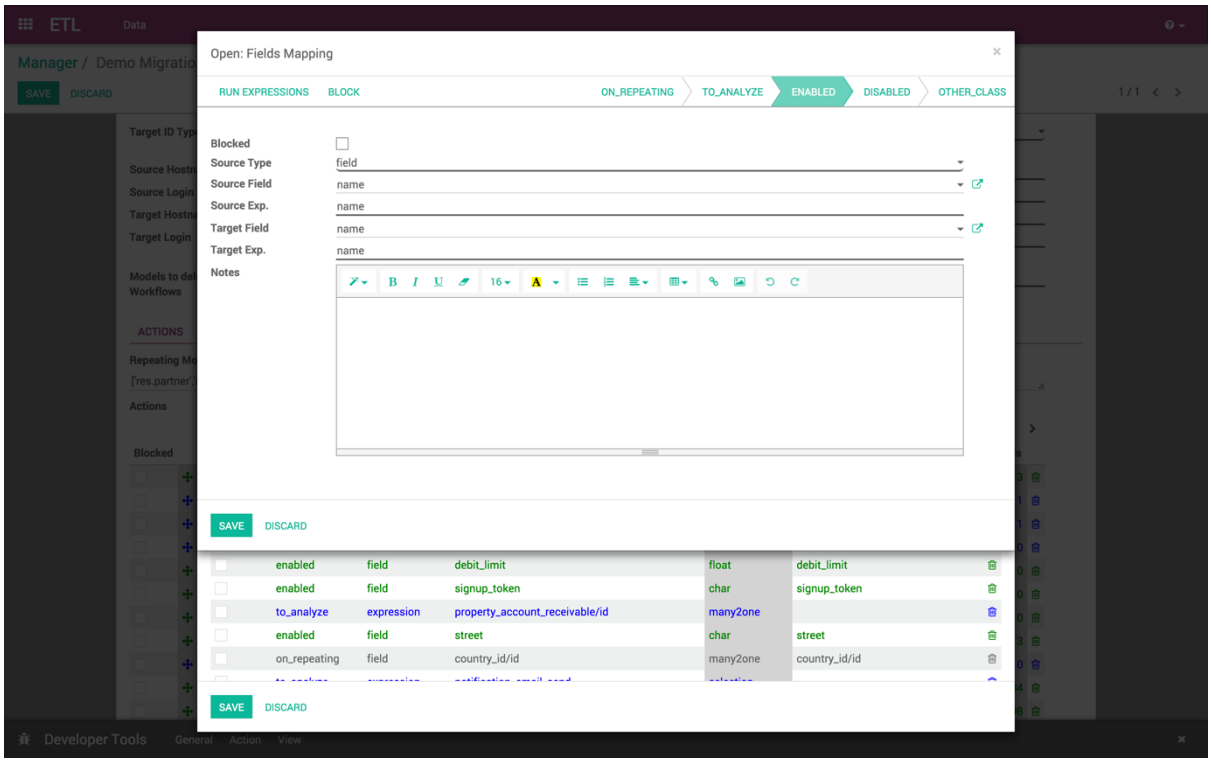
- *Name* field should be the name of the action which is usually automated from the *Match and Order* previous action.
- *Source Domain* field is used to apply domain for the source database model when performing the migration to filter out or include certain records in the migration.
- *Blocked* field is used to block the actions from running instead of having to switch the status to disabled. This field is used when configuring and testing the action on the first run of migration. After done configuring the action, *Blocked* field will usually be checked then later unchecked when performing the real migration which will be explained in the next section.
- *Sequence* field is used to order the action. The order for which action (model) will be performed first is really important due to the dependencies between models. For example, the sequence of customer tags model should be lower than the customer model since migration of the customer model will require the existing records of tags when the field of *tag_ids* is enabled (field configuration will be explained in the next section).
- *Repeating Action* field is a read-only field which will be automatically checked when the one of the fields state in the action's *Field Mapping* list is set to *on_repeating*. When this field is checked, the *Run Repeated Action* button will appear in the action bar of the *Actions* model form.
- *From Record* field is also used to filter out or include records in the migration process. The records that will be migrated will start from the value set at this field. To disable this feature, simply leave it along with the *To Record* field to its default value 0 (zero).
- *To Record* field is also used to filter out or include records in the migration process. The records that will be migrated will end at the value set at this field. To disable this feature, simply leave it along with the *From Record* field to its default value 0 (zero).

- *Source Model* field contains selections of the source model name.
- *source_id_exp* field is the field name of the ID field in the source model. Usually is set at its default (*id*).
- *Source Records* is a read only field counting the number of records at the source database in relation to the selected source model. Number of non-active records will not be counted, but can still be included in migration by setting the domain *['|', ('active', '=', True), ('active', '=', False)]*
- *Target Model* field contains selections of the target model name which will be mapped to receive the records from the source model when running the action.
- *Target ID Type* field have the same function as the *Target ID Type* field of the manager model. The default value will follow the value set at the *Target ID Type* field of the manager model and can be changed in every action according to preference (not recommended).
- *Target Records* is a read only field counting the number of records at the destination/target database in relation to the selected target model. Number of non-active records will not be counted.
- *target_id_prefix* field will only appear when the Target ID Type field is set to Builded ID allowing the customization of the records XML id instead of using the default export external ID.

The *Action* fields are usually set correctly by the automatic Match and Order action. Beside configuring the fields, it's very important to set the action's state which can be changed to the following possible state:

- *Enabled* state should be set to an action that will be included in the migration process.
- *To Analyse* state should be set to an action that require a further analysis and testing. When an action is set to this state, it will not be included when running the migration process.
- *Disabled* state should be set to an action that will not be included in the migration process.
- *No Records* state should be set to an action that will not be included in the migration process due to 0 records found in the source model.

After correctly configuring and checking the *Action* fields, it's very important to also check and configure every line of field mapping in the field mapping list in every actions. The field mapping determines which field of the selected model to be included or excluded in the migration process. To configure the fields, simply click the field mapping from the field mapping list of the action form.



Following is the details about the default fields in the field mapping model:

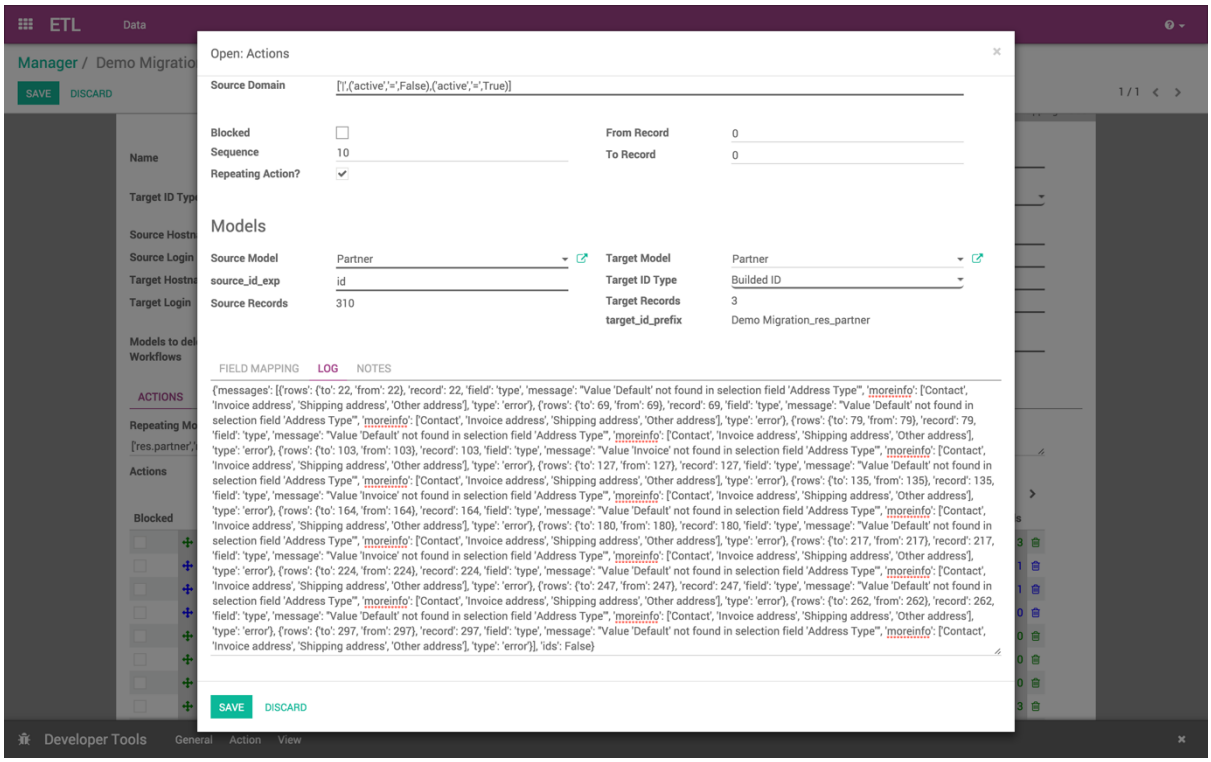
- *Blocked* field works in a similar way with the *Blocked* field of the action model which in this case is used to block fields that have been analysed so that the data of this field is not included when running the testing.
- *Source Type* is a selection field which is set to the default value *field* for standard field data migration. Other source types will be explained in the next section.
- *Source Field* is the selection of source field names in respect to the selected source model in the action model form.
- *Source Exp.* is a short description of the selected source field.
- *Target Field* is the selection of target field names in respect to the selected target model in the action model form.
- *Target Exp.* is a short description of the selected target field.
- *Notes* is a field prepared for ETL users to write a longer notes for each field mapping.

Field mapping also have a state similar to that of actions and it's also very important to set the field mapping's state which can be changed to the following possible state:

- *Enabled* state should be set to a field mapping that will be included in the action testing process and or migration process.
- *To Analyse* state should be set to a field mapping that require a further analysis and testing. When a field mapping is set to this state, it will not be included when running the action testing process and or migration process.
- *Disabled* state should be set to a field mapping that will not be included in the action testing process and or migration process.
- *Other Class* state should be set to a field mapping that involves a relational field in which the record will be migrated from the other model.

- *On Repeating* state should be set to a field mapping that usually involves a relational field that requires its own records such as parent/child relation or records from other models having a higher action sequence. This field data will be migrated after the first migration iteration by clicking *Run Repeated Action* in terms of action testing or *Run Repeated Actions* while running migration from the manager later after completing the action testing.

After configuring all the field mapping and the configuration for a specific action, test the action by clicking the *Run Action* button at the action bar for that specific action.



To see the result of the test, check the value of *Target Records*. If it increases after the process in regards to the *Source Records*, then the migration can be considered to be successful. To further confirm and check for errors, click the *Log* tab of the *Actions* form. When the test is successful, the logs will show an array of new created database id for the migrated records; otherwise, error messages will be shown. Address the error by reconfiguring the setting and field mappings of that specific action then re-run the test.

Step 11: Cleaning the Target and ETL Database

After correctly configuring and testing all the actions/model mapping, disable the other actions that will not be necessary for the intended migration and unblock all the actions. Before proceeding, do not forget to backup your ETL database.

Since the target database have been used for the testing, it's recommended to drop the database and recreate it. Make sure the modules are also installed again. If the target database name is changed, don't forget to change the Target Database at the ETL manager.

Step 12: Perform Migration

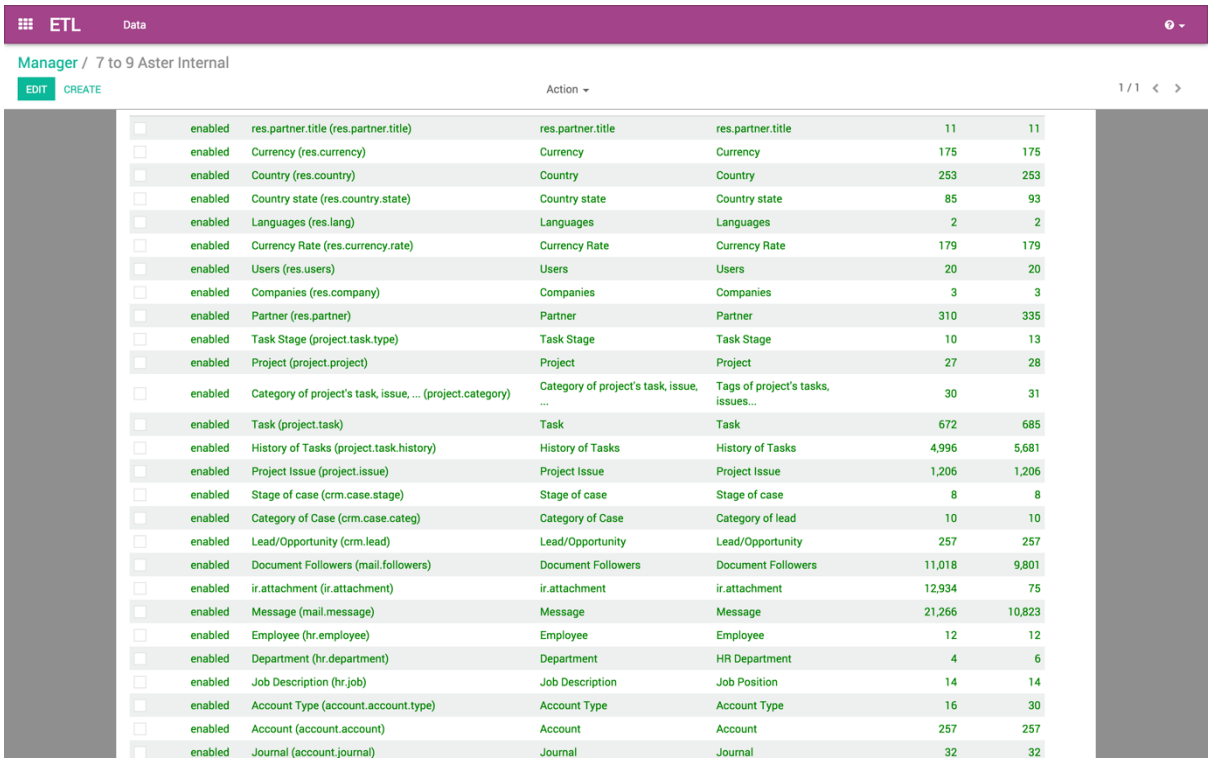
To perform the migration, simply click *Run Actions* button at the action bar of the manager form view. This will run all the actions according to our configuration in order. When process is completed, try checking for errors at every action's log since errors may still happen due to little misconfiguration.

When errors are found, try to address the errors accordingly by reconfiguring the fields then re-run the migration.

When no errors are found, click *Run Repeated Actions* button at the action bar of the manager form view as well to migrate the field mappings where state is set to *On Repeating*.

Re-check for error at the action logs and try to address them if there is one or more. After addressing the error, re-run the *Run Repeated Actions* action.

When no errors are found, migration can be considered to be successful.



enabled	res.partner.title (res.partner.title)	res.partner.title	res.partner.title	11	11
enabled	Currency (res.currency)	Currency	Currency	175	175
enabled	Country (res.country)	Country	Country	253	253
enabled	Country state (res.country.state)	Country state	Country state	85	93
enabled	Languages (res.lang)	Languages	Languages	2	2
enabled	Currency Rate (res.currency.rate)	Currency Rate	Currency Rate	179	179
enabled	Users (res.users)	Users	Users	20	20
enabled	Companies (res.company)	Companies	Companies	3	3
enabled	Partner (res.partner)	Partner	Partner	310	335
enabled	Task Stage (project.task.type)	Task Stage	Task Stage	10	13
enabled	Project (project.project)	Project	Project	27	28
enabled	Category of project's task, issue, ... (project.category)	Category of project's task, issue, ...	Tags of project's tasks, issues...	30	31
enabled	Task (project.task)	Task	Task	672	685
enabled	History of Tasks (project.task.history)	History of Tasks	History of Tasks	4,996	5,681
enabled	Project Issue (project.issue)	Project Issue	Project Issue	1,206	1,206
enabled	Stage of case (crm.case.stage)	Stage of case	Stage of case	8	8
enabled	Category of Case (crm.case.categ)	Category of Case	Category of lead	10	10
enabled	Lead/Opportunity (crm.lead)	Lead/Opportunity	Lead/Opportunity	257	257
enabled	Document Followers (mail.followers)	Document Followers	Document Followers	11,018	9,801
enabled	ir.attachment (ir.attachment)	ir.attachment	ir.attachment	12,934	75
enabled	Message (mail.message)	Message	Message	21,266	10,823
enabled	Employee (hr.employee)	Employee	Employee	12	12
enabled	Department (hr.department)	Department	HR Department	4	6
enabled	Job Description (hr.job)	Job Description	Job Position	14	14
enabled	Account Type (account.account.type)	Account Type	Account Type	16	30
enabled	Account (account.account)	Account	Account	257	257
enabled	Journal (account.journal)	Journal	Journal	32	32

Manually Mapping

Manual mapping for both models and fields are possible when the automatic *Match and Order* action is inaccurate.

To manually map a model, navigate to the actions list view and create a new action/model mapping. Select the manager in the *Manager* field of the action then enter the detail of the action fields accordingly as described in Step 10 of the migration process. If the Source

Model and the Target Model selection is empty, make sure the Manager field is set to the correct manager that have perform the *Read and Get Records* action.

After creating the action, click *Add an item* at the *Field Mapping* tab of that specific action to create the field mapping. Enter the detail of the field mapping fields accordingly as described in Step 10 of the migration process.

Error Handling for Selection Fields and Value Mapping

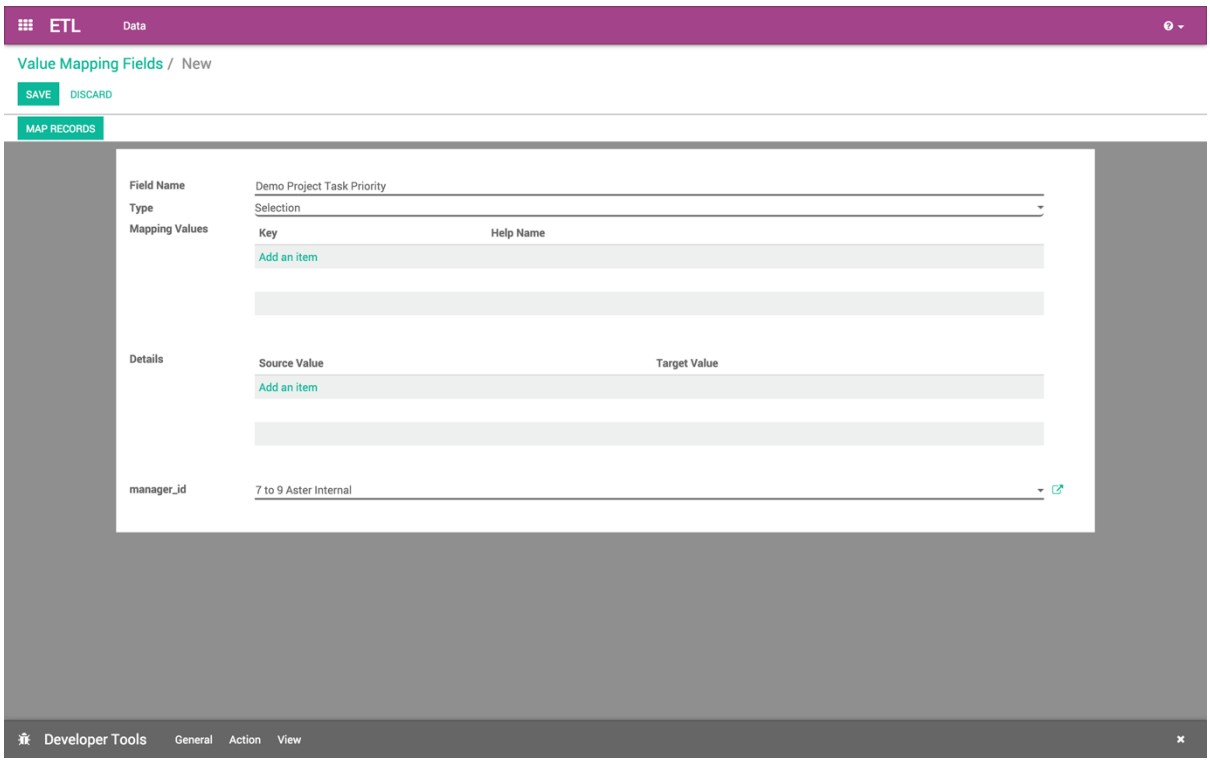
Selection fields may cause confusing errors during migration since the source field valid selection values may be different with the target field valid selection values.

For example, in OpenERP version 7.0, the *priority* field of the *project.task* model have the following selection range: “Very Low”, “Low”, “Medium”, “Important”, “Very Important”. In Odoo version 9.0, however, the selection range of the same field allows a different selection range such as following: “Normal”, “High”.

In this case, we need to utilize ETL’s Value Mapping Fields.

Value Mapping Fields

To use value mapping fields, navigate to the value mapping fields list view and click create.



Set a name to the value mapping field at the *Field Name* field then set the type value to *Selection*. Set the *manager_id* field value to the specific manager that will be used for the migration.

For every possible selection values (both at source and at destination), create a *Mapping Value* record by clicking *Add an item* at the *Mapping Values* list. *Key* should be the real selection value and *Help Name* can be a short description for that specific selection value or simply the same value with *Key*.

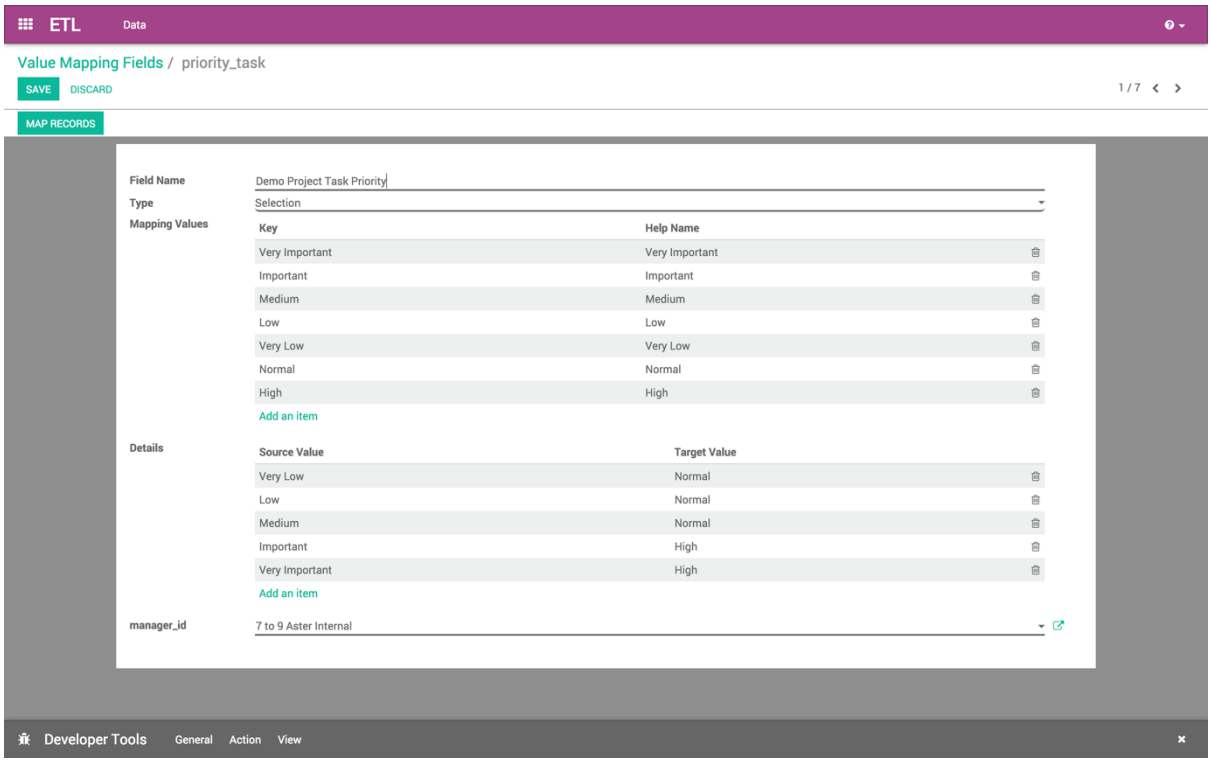
For example, the *Mapping Values* for the *project.task priority* field will be as following:

Key	Help Name
Very Important	Very Important
Important	Important
Medium	Medium
Low	Low
Very Low	Very Low
Normal	Normal
High	High

After setting the Mapping Values, do not directly do the Details list. Click save, then edit to continue entering the Details list. The value mapping will be done in the Details list according to the Source Value and Target Value.

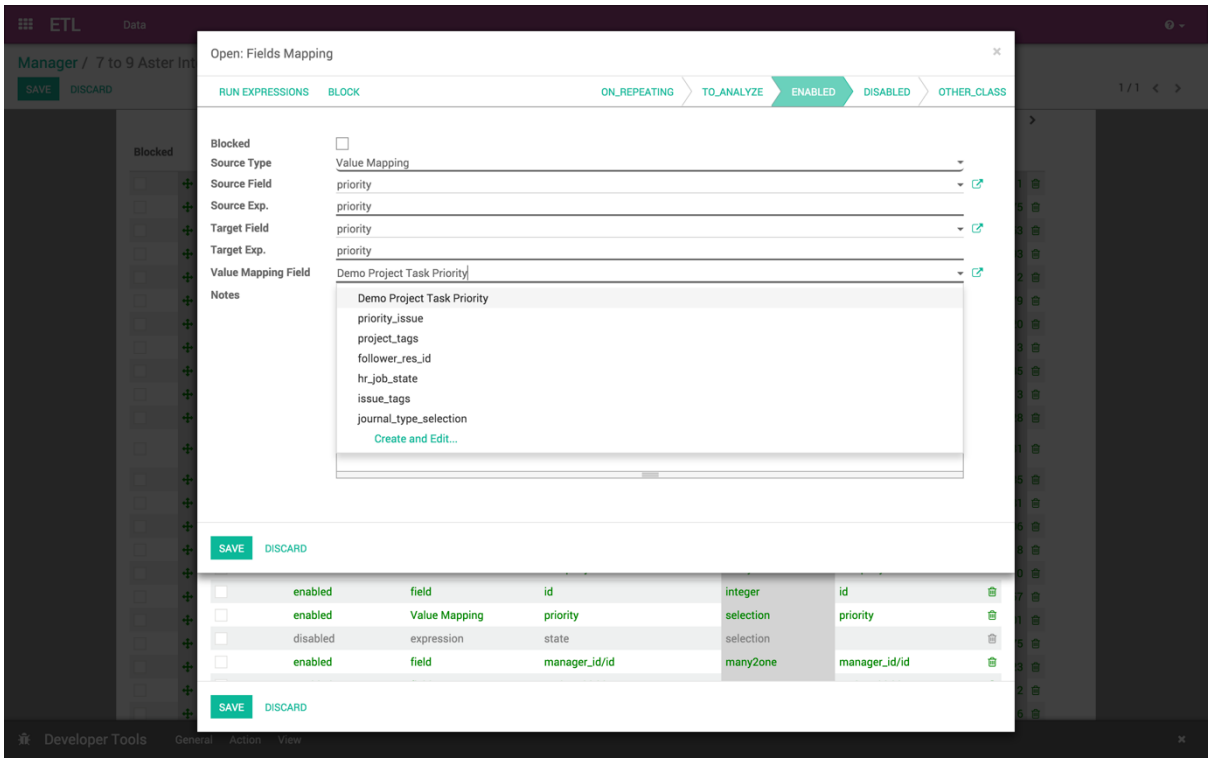
For example, the *Details* for the *project.task priority* field will be as following:

Source Value	Target Value
Very Low	Normal
Low	Normal
Medium	Normal
Important	High
Very Important	High



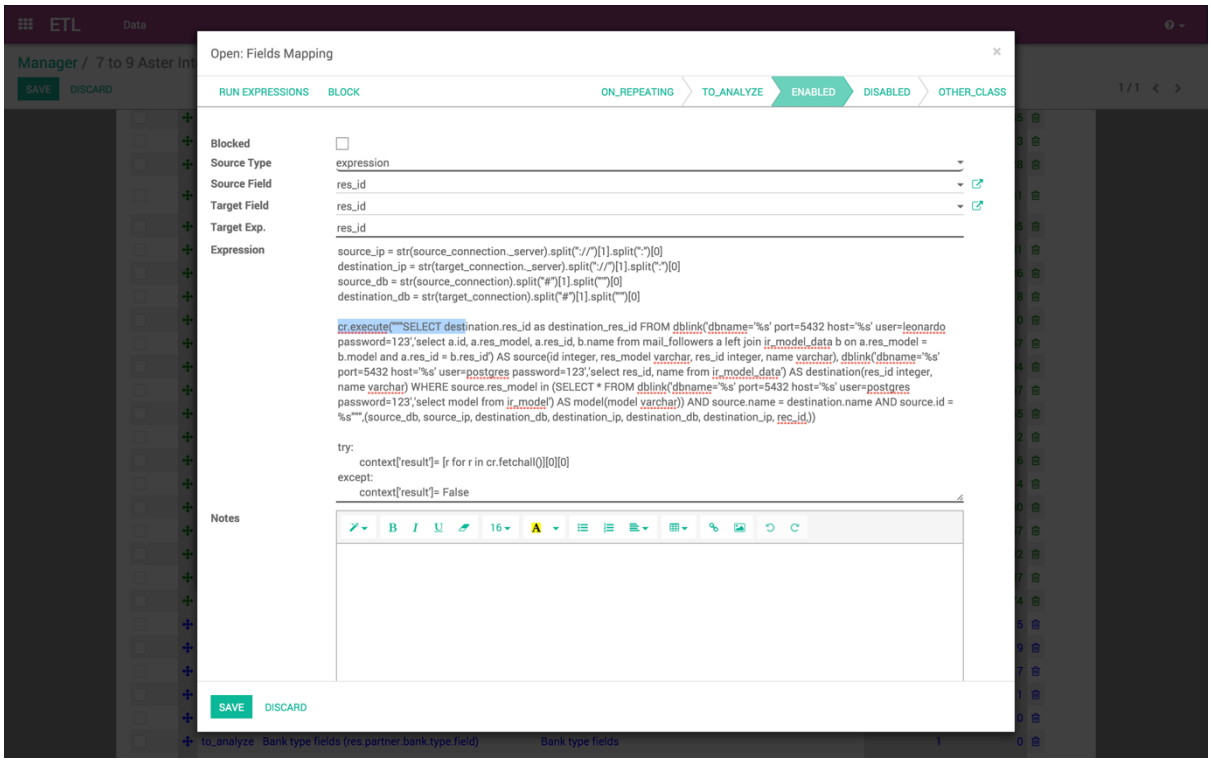
Click *Save* to save the *Value Mapping Fields* record.
When the Value Mapping Field for a specific selection field is have been created, navigate to the action containing that specific field mapping, click the intended field mapping, then set the *Source Type* field to *Value Mapping* and set the *Value Mapping Field* to the specific value mapping field record that have been created. Save the changes that have been made.

The value mapping example for the *project.task priority* selection field is shown according to the following image:



Python Expression

Some field mappings may be enhanced with python code to allow more dynamic values at the target field. To utilize expressions field mapping, navigate to the field mapping that will the require the expression then changing the *Source Type* into *expression*. After setting the type into expression, an additional field *expression* will appear. The python expression will be coded inside this field.



Following python code from *field_mapping.py* located inside the ETL addons directory shows possible objects that can be accessed from the expressions:

```
    @api.multi
    def run_expressions(
            self, rec_id, source_connection=False,
target_connection=False):
        result = []

        for field_mapping in self:
            expression_result = False
            if not source_connection or not target_connection:
                (source_connection, target_connection) =
field_mapping.action_id.manager_id.open_connections()
            source_model_obj = source_connection.model(
                field_mapping.action_id.source_model_id.model)
            target_model_obj = target_connection.model(
                field_mapping.action_id.target_model_id.model)

            obj_pool = source_model_obj
            cxt = {
                'self': obj_pool,   #to be replaced by target_obj
                'source_obj': source_model_obj,
                'source_connection': source_connection,
                'target_obj': target_model_obj,
                'target_connection': target_connection,
                'rec_id': rec_id,
                'pool': self.pool,
                'time': time,
                'cr': self._cr,
                # copy context to prevent side-effects of eval
                'context': dict(self._context),
                'uid': self.env.user.id,
```



```
                'user': self.env.user,
            }
            if not field_mapping.expression:
                raise Warning(_(
                    'Warning. Type expression choosen buy not expression
set'))
            # nocopy allows to return 'action'
            eval(field_mapping.expression.strip(), cxt, mode="exec")
            if 'result' in cxt['context']:
                expression_result = cxt['context'].get('result')
            result.append(expression_result)
        return result
```

For further details, please open *field_mapping.py* at the addons folder of the ETL module.

Error Handling for Relational Field Using Raw Integer as ID

Some models such as *mail.followers* has a field such as *res_id* that stores the ID of the resource/record it attached to in a raw integer type (*int*) instead of relational type (*many2one* / *one2many* / *many2many*). Hence, when it's migrated, there will be no technical error but the *res_id* remains the resource ID of the source database which may change in the destination database. This error can be solved by using python expressions in the field mapping. Following is the python expressions used to solve this issue related to the *mail.followers res_id*:

```
source_ip = str(source_connection._server).split("://")[1].split(":")[0]
destination_ip =
str(target_connection._server).split("://")[1].split(":")[0]
source_db = str(source_connection).split("#")[1].split("'")[0]
destination_db = str(target_connection).split("#")[1].split("'")[0]

cr.execute("""SELECT destination.res_id as destination_res_id FROM
dblink('dbname='%s' port=5432 host='%s' user=leonardo
password=123','select a.id, a.res_model, a.res_id, b.name from
mail_followers a left join ir_model_data b on a.res_model = b.model and
a.res_id = b.res_id') AS source(id integer, res_model varchar, res_id
integer, name varchar), dblink('dbname='%s' port=5432 host='%s'
user=postgres password=123','select res_id, name from ir_model_data') AS
destination(res_id integer, name varchar) WHERE source.res_model in
(SELECT * FROM dblink('dbname='%s' port=5432 host='%s' user=postgres
password=123','select model from ir_model') AS model(model varchar)) AND
source.name = destination.name AND source.id = %s""",(source_db,
source_ip, destination_db, destination_ip, destination_db, destination_ip,
rec_id,))

try:
    context['result']= [r for r in cr.fetchall()][0][0]
except:
    context['result']= False
```

Do note that the above python code uses the *dblink* extension function from Postgres which require details such as database port, user, and password. In above case, source Postgres database have the following credential:

- DB User : *leonardo*
- DB Password : *123*
- DB Port : *5432*

In above case, the destination Postgres database have the following credential:

- DB User : *postgres*
- DB Password : *123*
- DB Port : *5432*

It is very crucial to execute the following SQL at the ETL's PostgreSQL database (not source or destination) before using the expressions containing the *dblink* Postgres function:

```
CREATE EXTENSION dblink;
```

Error Handling for Create Date Field

ETL does not support the migration of the create and write date for all the Odoo models. After running the migration, create and write date will be set to the migration date. It is in fact that this create or write date field can be ignored in some modules, but for some other modules it may be crucial. In that case it's necessary to manipulate the create and or write date with python expressions to allow the accurate migration for create and or write date. Following is the python expression used to solve the create date issue related to the *crm.lead* model in which create date is crucial:

```
source_ip = str(source_connection._server).split("://")[1].split(":")[0]
destination_ip =
str(target_connection._server).split("://")[1].split(":")[0]
source_db = str(source_connection).split("#")[1].split("'")[0]
destination_db = str(target_connection).split("#")[1].split("'")[0]

cr.execute("""SELECT destination.id, source.create_date FROM
dblink('dbname='%s' port=5432 host='%s' user=leonardo
password=123','SELECT a.id, b.name, a.create_date FROM crm_lead a,
ir_model_data b WHERE a.id = b.res_id and b.model = ''crm.lead''') AS
source(id integer, name varchar, create_date timestamp),
dblink('dbname='%s' port=5432 host='%s' user=postgres
password=123','SELECT a.id, b.name, a.create_date FROM crm_lead a,
ir_model_data b WHERE a.id = b.res_id and b.model = ''crm.lead''') AS
destination(id integer, name varchar, create_date timestamp) WHERE
source.name = destination.name AND source.id = %s""",(source_db,
source_ip, destination_db, destination_ip, rec_id,))

matching_record = [r for r in cr.fetchall()][0]
dest_id = matching_record[0]
create_date = matching_record[1]

cr.execute("""SELECT dblink_exec('dbname='%s' port=5432 host='%s'
user=postgres password=123','UPDATE crm_lead SET create_date = TIMESTAMP
'%s' WHERE id = %s')""",(destination_db, destination_ip, create_date,
dest_id))

context['result']= str(create_date)
```

Do note that the above python code uses the *dblink* extension function from Postgres which require details such as database port, user, and password. In above case, source Postgres database have the following credential:

- DB User : *leonardo*
- DB Password : *123*

- DB Port : *5432*

In above case, the destination Postgres database have the following credential:

- DB User : *postgres*
- DB Password : *123*
- DB Port : *5432*

It is very crucial to execute the following SQL at the ETL's PostgreSQL database (not source or destination) before using the expressions containing the *dblink* Postgres function:

```
CREATE EXTENSION dblink;
```

Error Handling for Many to Many Field Migration

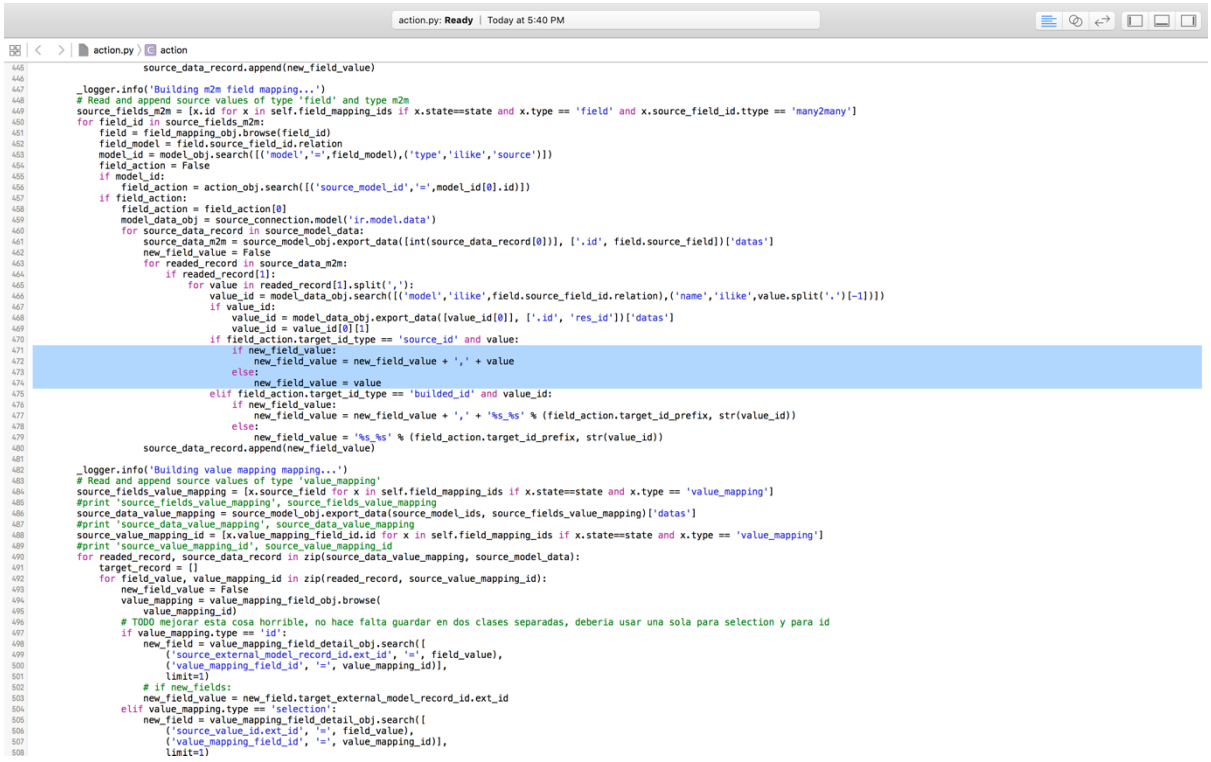
The ETL module source code contains a bug related to the migration of many to many field type. This can be solved by modifying the *action.py* python script at line **471** located at the ETL addons folder.

Replace:

```
new_field_value = value
```

Into:

```
if new_field_value:
      new_field_value = new_field_value + ',' + value
else:
      new_field_value = value
```



More about ETL's Migration Method

As mentioned earlier, one of the advantages of ETL is that it uses the native Odoo method. This can be found at the *action.py* python script at line **580** (unmodified *action.py*) located at the ETL addons folder.

action.py: **Ready** | Today at 5:55 PM

action.py 〉 run_action

```
552                     source_connection,
553                     target_connection)
554             rec.extend(migrated_id_results)
555
556         _logger.info('Building reference fields...')
557         field_mapping_reference_ids = [x.id for x in self.field_mapping_ids if x.state==state and x.type == 'reference']
558         if field_mapping_reference_ids:
559             for rec in source_model_data:
560                 rec_id = rec[0]
561                 reference_results = field_mapping_obj.browse(
562                     field_mapping_reference_ids).get_reference(
563                         int(rec_id), source_connection, target_connection)
564                 _logger.info('Reference_results: %s' % reference_results)
565                 rec.extend(reference_results)
566
567         _logger.info('Removing auxliaria .id')
568         target_model_data = []
569         #print source_model_data
570         #print ''
571         for record in source_model_data:
572             if self.target_id_type == 'source_id':
573                 target_model_data.append(record[1:])
574             elif self.target_id_type == 'builded_id':
575                 target_model_data.append(['%s_%s' % (
576                     self.target_id_prefix, str(record[0]))] + record[2:])
577
578         try:
579             _logger.info('Loadding Data...')
580             import_result = target_model_obj.load(
581                 target_fields, target_model_data)
582             vals = {'log': import_result}
583         except:
584             error = sys.exc_info()
585             print error
586             vals = {'log': error}
587
588         self.write(vals)
589         self.target_model_id.get_records(target_connection)
590
591     @api.multi
592     def order_actions(self, exceptions=None):
593         _logger.info('Lines to order %i' % len(self.ids))
594         if exceptions is None:
595             exceptions = []
596         # field_mapping_obj = self.pool.get('etl.field_mapping')
597         ordered_actions = []
598         ordered_ids = []
599
600         # We exclude de exceptions
601         unordered_ids = self.search([
602             ('id', 'in', self.ids),
603             ('source_model_id.model', 'not in', exceptions)]).ids
604         _logger.info('Request IDS: %s' % str(self.ids))
605         _logger.info('Request IDS without exceptions: %s' % str(unordered_ids))
606
607         actions_to_order = [
608             x.source_model_id.model for x in self.browse(unordered_ids)]
609         _logger.info('Actions_to_order %s' % actions_to_order)
610         count = 0
611         count_max = len(self) * 2
612         while unordered_ids and (count < count_max):
613             count += 1
614             rec = self.browse(unordered_ids[0])
615             action_clean_dependecies = []
```

ETL calls the load function of OpenERP to load the data into the target model. The load function can be found at the *models.py* python script starting at line **1022** (unmodified *models.py* at Odoo version 9) located at the OpenERP directory of Odoo.

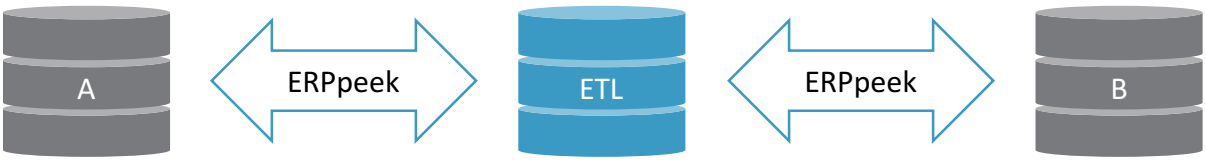
models.py: **Ready** | Today at 11:16 AM

models.py 〉 BaseModel

```python
        if context.get('defer_parent_store_computation'):
            self._parent_store_compute(cr)
        return position, 0, 0, 0

    def load(self, cr, uid, fields, data, context=None):
        """
        Attempts to load the data matrix, and returns a list of ids (or
        ``False`` if there was an error and no id could be generated) and a
        list of messages.

        The ids are those of the records created and saved (in database), in
        the same order they were extracted from the file. They can be passed
        directly to :meth:`~read`

        :param fields: list of fields to import, at the same index as the corresponding data
        :type fields: list(str)
        :param data: row-major matrix of data to import
        :type data: list(list(str))
        :param dict context:
        :returns: {ids: list(int)|False, messages: [Message]}
        """
        cr.execute('SAVEPOINT model_load')
        messages = []

        fields = map(fix_import_export_id_paths, fields)
        ModelData = self.pool['ir.model.data'].clear_caches()

        fg = self.fields_get(cr, uid, context=context)

        mode = 'init'
        current_module = ''
        noupdate = False

        ids = []
        for id, xid, record, info in self._convert_records(cr, uid,
                self._extract_records(cr, uid, fields, data,
                                      context=context, log=messages.append),
                context=context, log=messages.append):
            try:
                cr.execute('SAVEPOINT model_load_save')
            except psycopg2.InternalError, e:
                # broken transaction, exit and hope the source error was
                # already logged
                if not any(message['type'] == 'error' for message in messages):
                    messages.append(dict(info, type='error',message=
                        u"Unknown database error: '%s'" % e))
                break
            try:
                ids.append(ModelData._update(cr, uid, self._name,
                     current_module, record, mode=mode, xml_id=xid,
                     noupdate=noupdate, res_id=id, context=context))
                cr.execute('RELEASE SAVEPOINT model_load_save')
            except psycopg2.Warning, e:
                messages.append(dict(info, type='warning', message=str(e)))
                cr.execute('ROLLBACK TO SAVEPOINT model_load_save')
            except psycopg2.Error, e:
                messages.append(dict(
                    info, type='error',
                    **PGERROR_TO_OE[e.pgcode](self, fg, info, e)))
                # Failed to write, log to messages, rollback savepoint (to
                # avoid broken transaction) and keep going
                cr.execute('ROLLBACK TO SAVEPOINT model_load_save')
            except Exception, e:
```

About the ERPpeek

Every connection made from the ETL database to the source and target database uses the methods from python library called ERPpeek in which ERPpeek itself uses xmlrpc to communicate with the databases. The source and target destination is called as a class object *Client*. Actions done at those databases are also done using methods from ERPpeek.



The ERPpeek python code can be viewed at the following link: *https://github.com/tinyerp/erppeek/blob/master/erppeek.py*.

Change Log

Date	*By*	*Note*
27-Feb-2016	Leonardo Kurnia	Document Created