1.)
browse() Vs read() :-

Using read() is a bad practice. read() is used for web-services calls but in your own method calls you should always use browse(). Not only, it allows s a better quality of the code, but it's also better for the performance.

- read() calls name_get for many2one fields producing extra SQL queries you probably don't need.

- browse() is optimized for prefetching and auto-load of fields.

2.)
Def search([('field_name', 'operator', value)])
    Return :- recordset

Def search_read([domain], ['field_name'])
    Return :- list of dict
    [{'id': 3, 'name': u'Administrator'},
    {'id': 7, 'name': u'Agrolait'},]

3.)

Def copy(self, default):

    default: dictionary of field values to override in the original values of the copied record, e.g: {'field_name': overridden_value, ...}

    Return :- new record

def copy_data(self, default):

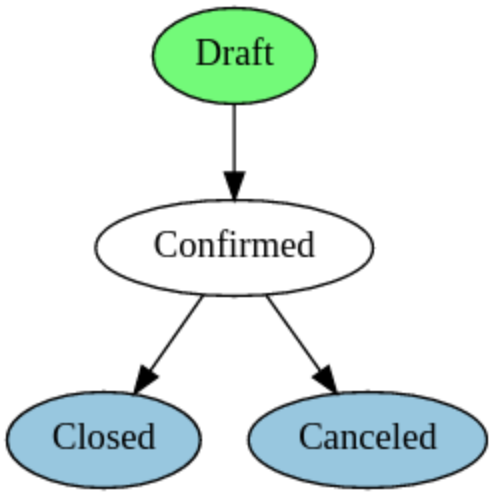
    default: field values to override in the original values of the copied record

    Return: dictionary containing all the field values

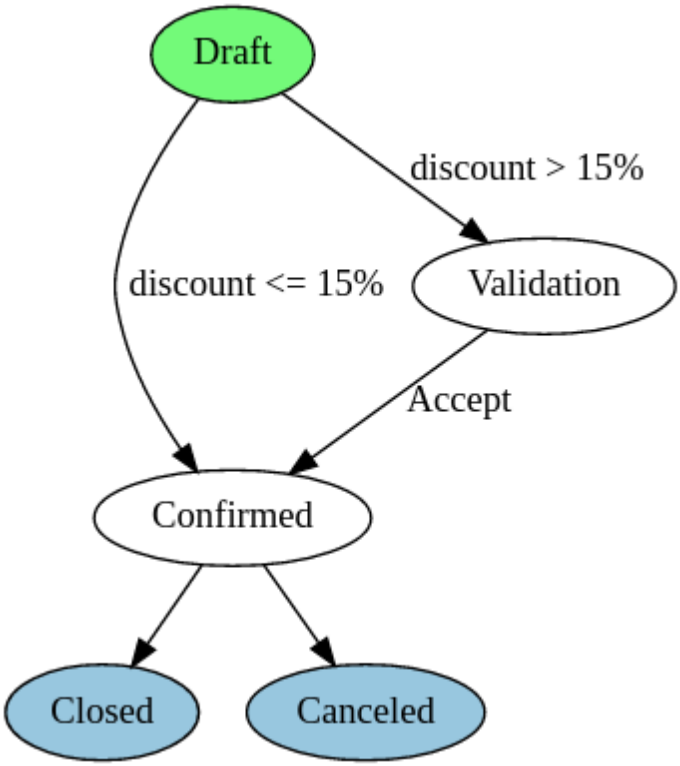
Worflow:-

Things to do with the associated record or model
The way of working on a record
Flow of working on a record
specifically, a workflow is a directed graph where **nodes** are called **Activities** and arcs are called **transitions**

- **Activities define work that should be done within the Odoo server, such as changing the state of some records**
- **Transitions control how the workflow progresses from activity to activity.**



Node, Activities:
Changing the state from draft to confirmed



Transitions:
When the state will change it will give 15% discount (by calling python function)

Workflow is written in xml file
Define workflow

<record id="wkf_sale" **model="workflow**">

<field **name="name">**sale.order.basic</field>
Can give any name

<field **name="osv">sale.order**</field>
on which model workflow will applied**(Model Name)**

<field **name="on_create">True**</field>
Because `on_create` is set to True on the workflow record, the workflow is instanciated for each newly created record.

(Otherwise, the workflow should be instanciated by other means, such as from some module Python code.)

</record>

Activity:

```
<record id="activity_a"  model="workflow.activity">
        <field name="wkf_id" ref="wkf_sale"/>
                        ref will have the id of workflow

        <field name="flow_start">True</field>
                        Odoo knows where to start the workflow

        <field name="name">a</field>

        <field name="kind">function</field>
                        That activity is of kind function, which means that the action print_a()
                        is a method call on the model test.workflow (the usual cr, uid,
                        ids, context arguments are passed for you)

         <field name="action">print_a()</field>
                        Python method
</record>

<record id="activity_b"  model="workflow.activity">
        <field name="wkf_id" ref="wkf_sale"/>
        <field name="flow_stop">True</field>
                        Odoo knows where to stop the workflow

         <field name="name">b</field>
         <field name="kind">function</field>
         <field name="action">print_b()</field>
</record>
```

Transistion:

```
<record id="trans_a_b" model="workflow.transition">
          <field name="act_from" ref="activity_a"/>
          <field name="act_to" ref="activity_b"/>
</record>
```

Kind:

An activity's kind defines the type of work an activity can perform.

- **Dummy (`dummy`, default)**

  Do nothing at all, or call a server action. Often used as dispatch or gather "hubs" for transitions.

- **Function (`function`)**

  Run some python code, execute a server action.

- **Stop all (`stopall`)**

  Completely stops the workflow instance and marks it as completed.

- **Subflow (`subflow`)**

  Starts executing an other workflow, once that workflow is completed the activity is done processing.

  By default, the subflow is instanciated for the same record as the parent workflow. It is possible to change that behavior by providing Python code that returns a record ID (of the same data model as the subflow). The embedded subflow instance is then the one of the given record.

Transitions:

When an activity is completed, the workflow engine tries to get depart from the completed activity, towards the next activities
In their simplest form (as in the example above), they link activities sequentially: activities are processed as soon as the activities preceding them are completed.

Instead of running all activities in one fell swoop, it is also possible to wait on transitions, going through them only when some criteria are met. The criteria are the **conditions**, the **signals**, and the **triggers**.

1.)Conditions:-

<field name="**condition**">procurement_needed() and ((order_policy!='prepaid') or invoiced)</field>

If the condition is not met, it will be reevaluated every time the associated record is modified, or by an explicit method call to do it.
By default, the attribute `condition` (i.e., the expression to be evaluated) is just "True"

2.)**Signal:-**

<field name="signal">ship_except</field>
<field name="condition" eval="True"/> <!-- Force empty →

In addition to a condition, a transition can specify a signal name. When such a signal name is present, the transition is not taken directly, even if the condition evaluates to `True`. Instead the transition blocks, waiting to be woken up.

A common way to send a signal is to use a button in the user interface, using the element `<button/>`

```
Source Activity :re-open

Destination Activity :cancel

Signal (Button Name) :invoice_cancel

Condition :True
```

Means in signal it will display a button till source to destination activity

3.)Trigger:-

```
<field name="signal">ship_end</field>
<field name="trigger_model" eval="False"/> <!-- Force empty  →
<field name="trigger_expr_id" eval="False"/> <!-- Force empty -->
<field name="condition" eval="True"/> <!-- Force empty →
```

With conditions that evaluate to `False`, transitions are not taken (and thus the activity it leads to is not processed immediately). Still, the workflow instance can get new chances to progress across that transition by providing so-called triggers. The idea is that when the condition is not satisfied, triggers are recorded in database. Later, it is possible to wake up specifically the workflow instances that installed those triggers, offering them to reevaluate their transition conditions.

Note
triggers are not re-installed whenever the transition is re-tried

Security:

1.) Access control:-

The permission of to perform the operation like read, edit, delete, create

Basically it is given through **ir.model.access.csv** file

model_purchase_team,purchase.group_purchase_user,1,0,0,0

model_purchase_team,purchase.group_purchase_manager,1,1,1,0

Groups are assign to user and permission are given to group

Groups are written in xml file with model=”res.groups”

<field name="category_id"

<field name="implied_ids"

2.) Record rule :-

Record rules are use to display the restricted records

For ex:- a particluar user should be able to particluar records only

Record rule are written in xml file with model=”ir.rule”

<record id="purchase_memeber_user_purchase_order_records" **model="ir.rule"**>

<field name="name">Only login user purchase records</field>

<field name="**model_id**" ref="**model_purchase_orde**r"/>

a model on which it applies

<field name="**groups**" eval="[(4, ref(**'purchase.group_purchase_use**r'))]"/>

a set of user groups to which the rule applies, if no group is specified the rule is *global*

<field name="**domain_force**">[('**purchase_member','=',user.id)**]</field>

The purchase_member is the filed

</record>

<record id="purchase_memeber_manager_purchase_order_records" model="ir.rule">

<field name="name">Only login manager purchase records</field>

<field name="model_id" ref="model_purchase_order"/>

<field name="groups" eval="[(4, ref('**purchase.group_purchase_manager**'))]"/>

<field name="**domain_force">[(1, '=', 1)**]</field>

</record>

Note:-

record rules do not apply to the Administrator user

although access rules do

3.) Field Access:-

An ORM `Field` can have a `groups` attribute providing a list of groups (as a comma-separated string of external identifiers).

If the current user is not in one of the listed groups, he will not have access to the field:

- restricted fields are automatically removed from requested views
- restricted fields are removed from `fields_get()` responses
- attempts to (explicitly) read from or write to restricted fields results in an access error

Field access can be given through xml file or py file

Example:-

Python file

standard_price = fields.Float(string='Cost Price', digits=dp.get_precision('Product Price'), groups="base.group_user", company_dependent=True)

Xml file

<field name="centralized" groups="account.group_account_manager"/>

4.) workflow transition rules:-

Workflow transitions can be restricted to a specific group. Users outside the group can not trigger the transition

********** In detail explation is not given so it will be done later **********

Actions:- used to present visualisations of a model through views

Their are **5** type of actions:

1.) **ir.actions.act_window**

For instance, to open customers (partner with the `customer` flag set) with list and form views:

```
{
    "type": "ir.actions.act_window",
    "res_model": "res.partner",
    "views": [[False, "tree"], [False, "form"]],
    "domain": [["customer", "=", true]],
}
```

Or to open the form view of a specific product (obtained separately) in a new dialog:

```
{
    "type": "ir.actions.act_window",
    "res_model": "product.product",
    "views": [[False, "form"]],
    "res_id": a_product_id,
    "target": "new",
}
```

2.) ir.actions.act_url

Allow opening a URL (website/web page) via an Odoo action. Can be customized via two fields:

Url :-the address to open when activating the action.

Traget :-opens the address in a new window/page if `new`, replaces the current content with the page if `self`. Defaults to `new`

```
{
    "type": "ir.actions.act_url",
    "url": "http://odoo.com",
    "target": "self",
}
```

will replace the current content section by the Odoo home page

3.) ir.actions.report.xml

Triggers the printing of a report

4.) Server Actions (ir.actions.server)

Allow triggering complex server code from any valid action location

@Decorator:- It is a function that take another function and extends the behaviour of latter function

Recordsets are immutable

Set operations:-

- `set1 | set2` returns the union of the two recordsets, a new recordset containing all records present in either source(**eliminate duplicate**)
- `set1 & set2` returns the intersection of two recordsets, a new recordset containing only records present in both sources(**return duplicate**)
- `set1 - set2` returns a new recordset containing only records of `set1` which are *not* in `set2`(**`it will eliminate the duplicate and return the remaining value of set1`**)

Recordset operations:-

(`map()`, `sorted()`, `ifilter()`, ...) return either a `list`

`filtered()`:

returns a recordset containing only records satisfying the provided predicate function. The predicate can also be a string to filter by a field being true or false
Example:-
only keep records whose company is the current user's
records.**filtered**(lambda r: r.company_id == user.company_id)

only keep records whose partner is a company
records.**filtered**("partner_id.is_company")

`sorted()`

returns a recordset sorted by the provided **key function**. If no key is provided, use the model's default sort order:

#sort records by name

records.sorted(key=lambda r: r.name)

`mapped()`

applies the provided function to each record in the recordset, returns a recordset if the results are recordsets:

```
# returns a list of summing two fields for each record in the set
    records.mapped(lambda r: r.field1 + r.field2)
```

The provided function can be a string to get field values:

```
# returns a list of names
    records.mapped('name')
# returns a recordset of partners
    record.mapped('partner_id')
# returns the union of all partner banks, with duplicates removed
    record.mapped('partner_id.bank_ids')
```

Company-dependent fields (property fields)

Formerly known as 'property' fields, the value of those fields depends on the company. In other words, users that belong to different companies may see different values for the field on a given record.

fields.property(

Report:-

In xml file

attachment_use:-

if set to True, the report will be stored as an attachment of the record using the name generated by the attachment expression; you can use this if you need your report to be generated only once (for legal reasons, for example)

attachment:-

python expression that defines the name of the report; the record is acessible as the variable object

Mandatory

report_view.xml(file 1)

1)

<**report id**="ob_sunrise_sale_report"

model="amazon.sale.order.ept" string="Sales Report"

report_type="**qweb-pdf / qweb-html"**

name="ob_sunrise_report.**sales_report_template**"
file="ob_sunrise_report.**sales_report_template**" />

attachment_use="True"

attachment="(object.state in ('open','paid')) and

('INV'+(object.number or '').replace('/','')+'.pdf')"

/>

2)

<record id="ob_sunrise_sale_report"

model="ir.actions.report.xml">

<field name="paperformat_id" ref="ob_sunrise_report.paperformat_sunrise_sales_report" />

</record>

3)

<record id="paperformat_sunrise_sales_report" model="**report.paperformat**">

```
        <field name="name">Sales Report</field>
        <field name="default" eval="True" />
        <field name="format">A4</field>
        <field name="orientation">Portrait</field>
        <field name="margin_top">40</field>
        <field name="margin_bottom">5</field>
        <field name="margin_left">3</field>
        <field name="margin_right">3</field>
        <field name="header_line" eval="False" />
        <field name="header_spacing">35</field>
        <field name="dpi">90</field>
    </record>
```

template_view.xml(file 2)

```
openerp>
  <data>
    <template id="sale_order_report_custom">
      <t t-call="report.external_layout">
        <div class="page">
   </template>
  <template id="sales_report_template">
      <t t-call="report.html_container">
      <t t-foreach="all_order" t-as="orderset">
      <t t-call="ob_sunrise_report.sale_order_report_custom"/>
 </template>
```

Python file

```
class sales_report(report_sxw.rml_parse):
  _name = 'sunrise.sales.report'
```

```
    def __init__(self, cr, uid, name, context):
        super(sales_report, self).__init__(cr, uid, name, context=context)
self.localcontext.update({
            'all_order': orders or [],
            'time': time,
            'Methods_1 ': self.methods_1})
Def methods_1()
Def methods_2()


class report_sales_new(models.AbstractModel):
    _name = 'report.ob_sunrise_report.sales_report_template'
    _inherit = 'report.abstract_report'
    _template = 'ob_sunrise_report.sales_report_template'
    _wrapped_report_class = sales_report
```

Customize existing report:

```
<template id="report_quotation_order_inherit_new"
inherit_id="sale.report_saleorder_document">
            <xpath expr="//table" position="replace">
```