

```

1  # sale_order.py
2  # Part of Odoo. See LICENSE file for full copyright and licensing details.
3  from collections import defaultdict
4  from datetime import timedelta
5  from itertools import groupby
6
7  from odoo import api, fields, models, SUPERUSER_ID, _
8  from odoo.exceptions import AccessError, UserError, ValidationError
9  from odoo.fields import Command
10 from odoo.osv import expression
11 from odoo.tools import float_is_zero, format_amount, format_date, html_keep_url,
12 is_html_empty
13 from odoo.tools.sql import create_index
14
15
16 from odoo.addons.payment import utils as payment_utils
17
18 INVOICE_STATUS = [
19     ('upselling', 'Upselling Opportunity'),
20     ('invoiced', 'Fully Invoiced'),
21     ('to invoice', 'To Invoice'),
22     ('no', 'Nothing to Invoice')
23 ]
24
25 SALE_ORDER_STATE = [
26     ('draft', "Quotation"),
27     ('sent', "Quotation Sent"),
28     ('sale', "Sales Order"),
29     ('cancel', "Cancelled"),
30 ]
31
32 class SaleOrder(models.Model):
33     _name = 'sale.order'
34     _inherit = ['portal.mixin', 'product.catalog.mixin', 'mail.thread',
35                 'mail.activity.mixin', 'utm.mixin']
36     _description = "Sales Order"
37     _order = 'date_order desc, id desc'
38     _check_company_auto = True
39
40     _sql_constraints = [
41         ('date_order_conditional_required',
42          "CHECK((state = 'sale' AND date_order IS NOT NULL) OR state != 'sale')",
43          "A confirmed sales order requires a confirmation date."),
44     ]
45
46 @property
47 def _rec_names_search(self):
48     if self._context.get('sale_show_partner_name'):
49         return ['name', 'partner_id.name']
50     return ['name']
51
52 #=== FIELDS ===#
53
54 name = fields.Char(
55     string="Order Reference",
56     required=True, copy=False, readonly=False,
57     index='trigram',
58     default=lambda self: _('New'))
59
60 company_id = fields.Many2one(
61     comodel_name='res.company',
62     required=True, index=True,
63     default=lambda self: self.env.company)
64
65 partner_id = fields.Many2one(
66     comodel_name='res.partner',
67     string="Customer",
68     required=True, change_default=True, index=True,
69     tracking=1,
70     domain="[('company_id', 'in', (False, company_id))]",
71     state = fields.Selection(
72         selection=SALE_ORDER_STATE,
73         string="Status",

```

```

71         readonly=True, copy=False, index=True,
72         tracking=3,
73         default='draft')
74     locked = fields.Boolean(default=False, copy=False, help="Locked orders cannot be
modified.")
75
76     client_order_ref = fields.Char(string="Customer Reference", copy=False)
77     create_date = fields.Datetime( # Override of default create_date field from ORM
78         string="Creation Date", index=True, readonly=True)
79     commitment_date = fields.Datetime(
80         string="Delivery Date", copy=False,
81         help="This is the delivery date promised to the customer. "
82             "If set, the delivery order will be scheduled based on "
83             "this date rather than product lead times.")
84     date_order = fields.Datetime(
85         string="Order Date",
86         required=True, copy=False,
87         help="Creation date of draft/sent orders,\nConfirmation date of confirmed
orders.",
88         default=fields.Datetime.now)
89     origin = fields.Char(
90         string="Source Document",
91         help="Reference of the document that generated this sales order request")
92     reference = fields.Char(
93         string="Payment Ref.",
94         help="The payment communication of this sale order.",
95         copy=False)
96
97     require_signature = fields.Boolean(
98         string="Online signature",
99         compute='_compute_require_signature',
100        store=True, readonly=False, precompute=True,
101        help="Request a online signature from the customer to confirm the order.")
102     require_payment = fields.Boolean(
103         string="Online payment",
104         compute='_compute_require_payment',
105         store=True, readonly=False, precompute=True,
106         help="Request a online payment from the customer to confirm the order.")
107     prepayment_percent = fields.Float(
108         string="Prepayment percentage",
109         compute='_compute_prepayment_percent',
110         store=True, readonly=False, precompute=True,
111         help="The percentage of the amount needed that must be paid by the customer
to confirm the order.")
112
113     signature = fields.Image(
114         string="Signature",
115         copy=False, attachment=True, max_width=1024, max_height=1024)
116     signed_by = fields.Char(
117         string="Signed By", copy=False)
118     signed_on = fields.Datetime(
119         string="Signed On", copy=False)
120
121     validity_date = fields.Date(
122         string="Expiration",
123         compute='_compute_validity_date',
124         store=True, readonly=False, copy=False, precompute=True)
125     journal_id = fields.Many2one(
126         'account.journal', string="Invoicing Journal",
127         compute='_compute_journal_id', store=True, readonly=False, precompute=True,
128         domain=[('type', '=', 'sale')], check_company=True,
129         help="If set, the SO will invoice in this journal; "
130             "otherwise the sales journal with the lowest sequence is used.")
131
132     # Partner-based computes
133     note = fields.Html(
134         string="Terms and conditions",
135         compute='_compute_note',
136         store=True, readonly=False, precompute=True)
137
138     partner_invoice_id = fields.Many2one(
139         comodel_name='res.partner',

```

```

140         string="Invoice Address",
141         compute='_compute_partner_invoice_id',
142         store=True, readonly=False, required=True, precompute=True,
143         domain="[('company_id', '=', False), ('company_id', '=', company_id)]")
144 partner_shipping_id = fields.Many2one(
145     comodel_name='res.partner',
146     string="Delivery Address",
147     compute='_compute_partner_shipping_id',
148     store=True, readonly=False, required=True, precompute=True,
149     domain="[('company_id', '=', False), ('company_id', '=', company_id)]"),
150
151 fiscal_position_id = fields.Many2one(
152     comodel_name='account.fiscal.position',
153     string="Fiscal Position",
154     compute='_compute_fiscal_position_id',
155     store=True, readonly=False, precompute=True, check_company=True,
156     help="Fiscal positions are used to adapt taxes and accounts for particular
157         customers or sales orders/invoices."
158     "The default value comes from the customer.",
159     domain="[('company_id', '=', company_id)]")
160 payment_term_id = fields.Many2one(
161     comodel_name='account.payment.term',
162     string="Payment Terms",
163     compute='_compute_payment_term_id',
164     store=True, readonly=False, precompute=True, check_company=True, #
165     Unrequired company
166     domain="[('company_id', '=', False), ('company_id', '=', company_id)]")
167 pricelist_id = fields.Many2one(
168     comodel_name='product.pricelist',
169     string="Pricelist",
170     compute='_compute_pricelist_id',
171     store=True, readonly=False, precompute=True, check_company=True, #
172     Unrequired company
173     tracking=1,
174     domain="[('company_id', '=', False), ('company_id', '=', company_id)]",
175     help="If you change the pricelist, only newly added lines will be affected.")
176 currency_id = fields.Many2one(
177     comodel_name='res.currency',
178     compute='_compute_currency_id',
179     store=True,
180     precompute=True,
181     ondelete='restrict'
182 )
183 currency_rate = fields.Float(
184     string="Currency Rate",
185     compute='_compute_currency_rate',
186     digits=(12, 6),
187     store=True, precompute=True)
188 user_id = fields.Many2one(
189     comodel_name='res.users',
190     string="Salesperson",
191     compute='_compute_user_id',
192     store=True, readonly=False, precompute=True, index=True,
193     tracking=2,
194     domain=lambda self: "[('groups_id', '=', {}), ('share', '=', False),
195         ('company_ids', '=', company_id)].format(
196             self.env.ref('sales_team.group_sale_salesman').id
197         )
198 )
199 team_id = fields.Many2one(
200     comodel_name='crm.team',
201     string="Sales Team",
202     compute='_compute_team_id',
203     store=True, readonly=False, precompute=True, ondelete="set null",
204     change_default=True, check_company=True, # Unrequired company
205     tracking=True,
206     domain="[('company_id', '=', False), ('company_id', '=', company_id)]")
207
208 # Lines and line based computes
209 order_line = fields.One2many(
210     comodel_name='sale.order.line',
211     inverse_name='order_id',
212     string="Order Lines",

```

```

208         copy=True, auto_join=True)
209
210     amount_untaxed = fields.Monetary(string="Untaxed Amount", store=True, compute=
211     '_compute_amounts', tracking=5)
212     amount_tax = fields.Monetary(string="Taxes", store=True, compute=
213     '_compute_amounts')
214     amount_total = fields.Monetary(string="Total", store=True, compute=
215     '_compute_amounts', tracking=4)
216     amount_to_invoice = fields.Monetary(string="Amount to invoice", store=True,
217     compute='_compute_amount_to_invoice')
218     amount_invoiced = fields.Monetary(string="Already invoiced", compute=
219     '_compute_amount_invoiced')
220
221     invoice_count = fields.Integer(string="Invoice Count", compute='_get_invoiced')
222     invoice_ids = fields.Many2many(
223         comodel_name='account.move',
224         string="Invoices",
225         compute='_get_invoiced',
226         search='_search_invoice_ids',
227         copy=False)
228     invoice_status = fields.Selection(
229         selection=INVOICE_STATUS,
230         string="Invoice Status",
231         compute='_compute_invoice_status',
232         store=True)
233
234     # Payment fields
235     transaction_ids = fields.Many2many(
236         comodel_name='payment.transaction',
237         relation='sale_order_transaction_rel', column1='sale_order_id', column2=
238         'transaction_id',
239         string="Transactions",
240         copy=False, readonly=True)
241     authorized_transaction_ids = fields.Many2many(
242         comodel_name='payment.transaction',
243         string="Authorized Transactions",
244         compute='_compute_authorized_transaction_ids',
245         copy=False,
246         compute_sudo=True)
247     amount_paid = fields.Float(compute='_compute_amount_paid', compute_sudo=True)
248
249     # UTMs - enforcing the fact that we want to 'set null' when relation is unlinked
250     campaign_id = fields.Many2one(onDelete='set null')
251     medium_id = fields.Many2one(onDelete='set null')
252     source_id = fields.Many2one(onDelete='set null')
253
254     # Followup ?
255     analytic_account_id = fields.Many2one(
256         comodel_name='account.analytic.account',
257         string="Analytic Account",
258         copy=False, check_company=True, # Unrequired company
259         domain="[('company_id', '=', False), ('company_id', '=', company_id)]")
260     tag_ids = fields.Many2many(
261         comodel_name='crm.tag',
262         relation='sale_order_tag_rel', column1='order_id', column2='tag_id',
263         string="Tags")
264
265     # Remaining non stored computed fields (hide/make fields readonly, ...)
266     amount_undiscounted = fields.Float(
267         string="Amount Before Discount",
268         compute='_compute_amount_undiscounted', digits=0)
269     country_code = fields.Char(related='company_id.account_fiscal_country_id.code',
270     string="Country code")
271     expected_date = fields.Datetime(
272         string="Expected Date",
273         compute='_compute_expected_date', store=False, # Note: can not be stored
274         since depends on today()
275         help="Delivery date you can promise to the customer, computed from the
276         minimum lead time of the order lines.")
277     is_expired = fields.Boolean(string="Is Expired", compute='_compute_is_expired')
278     partner_credit_warning = fields.Text(
279         compute='_compute_partner_credit_warning')

```

```

271 tax_calculation_rounding_method = fields.Selection(
272     related='company_id.tax_calculation_rounding_method',
273     depends=['company_id'])
274 tax_country_id = fields.Many2one(
275     comodel_name='res.country',
276     compute='_compute_tax_country_id',
277     # Avoid access error on fiscal position when reading a sale order with
278     # company != user.company_ids
279     compute_sudo=True) # used to filter available taxes depending on the fiscal
280     country and position
281 tax_totals = fields.Binary(compute='_compute_tax_totals', exportable=False)
282 terms_type = fields.Selection(related='company_id.terms_type')
283 type_name = fields.Char(string="Type Name", compute='_compute_type_name')
284
285 # Remaining ux fields (not computed, not stored)
286
287 show_update_fpos = fields.Boolean(
288     string="Has Fiscal Position Changed", store=False) # True if the fiscal
289     position was changed
290 has_active_pricelist = fields.Boolean(
291     compute='_compute_has_active_pricelist')
292 show_update_pricelist = fields.Boolean(
293     string="Has Pricelist Changed", store=False) # True if the pricelist was
294     changed
295
296 def init(self):
297     create_index(self._cr, 'sale_order_date_order_id_idx', 'sale_order', [
298         "date_order desc", "id desc"])
299
300 #=== COMPUTE METHODS ===#
301
302 @api.depends('partner_id')
303 @api.depends_context('sale_show_partner_name')
304 def _compute_display_name(self):
305     if not self._context.get('sale_show_partner_name'):
306         return super()._compute_display_name()
307     for order in self:
308         name = order.name
309         if order.partner_id.name:
310             name = f'{name} - {order.partner_id.name}'
311         order.display_name = name
312
313 @api.depends('company_id')
314 def _compute_require_signature(self):
315     for order in self:
316         order.require_signature = order.company_id.portal_confirmation_sign
317
318 @api.depends('company_id')
319 def _compute_require_payment(self):
320     for order in self:
321         order.require_payment = order.company_id.portal_confirmation_pay
322
323 @api.depends('require_payment')
324 def _compute_prepayment_percent(self):
325     for order in self:
326         order.prepayment_percent = order.company_id.prepayment_percent
327
328 @api.depends('company_id')
329 def _compute_validity_date(self):
330     today = fields.Date.context_today(self)
331     for order in self:
332         days = order.company_id.quotation_validity_days
333         if days > 0:
334             order.validity_date = today + timedelta(days)
335         else:
336             order.validity_date = False
337
338 def _compute_journal_id(self):
339     self.journal_id = False
340
341 @api.depends('partner_id')
342 def _compute_note(self):

```

```

338     use_invoice_terms = self.env['ir.config_parameter'].sudo().get_param(
339         'account.use_invoice_terms')
340     if not use_invoice_terms:
341         return
342     for order in self:
343         order = order.with_company(order.company_id)
344         if order.terms_type == 'html' and self.env.company.invoice_terms_html:
345             baseurl = html_keep_url(order._get_note_url() + '/terms')
346             context = {'lang': order.partner_id.lang or self.env.user.lang}
347             order.note = _('Terms & Conditions: %s', baseurl)
348             del context
349         elif not is_html_empty(self.env.company.invoice_terms):
350             order.note = order.with_context(lang=order.partner_id.lang).env.
351                 company.invoice_terms
352
353     @api.model
354     def _get_note_url(self):
355         return self.env.company.get_base_url()
356
357     @api.depends('partner_id')
358     def _compute_partner_invoice_id(self):
359         for order in self:
360             order.partner_invoice_id = order.partner_id.address_get(['invoice'])[
361                 'invoice'] if order.partner_id else False
362
363     @api.depends('partner_id')
364     def _compute_partner_shipping_id(self):
365         for order in self:
366             order.partner_shipping_id = order.partner_id.address_get(['delivery'])[
367                 'delivery'] if order.partner_id else False
368
369     @api.depends('partner_shipping_id', 'partner_id', 'company_id')
370     def _compute_fiscal_position_id(self):
371         """
372         Trigger the change of fiscal position when the shipping address is modified.
373         """
374         cache = {}
375         for order in self:
376             if not order.partner_id:
377                 order.fiscal_position_id = False
378                 continue
379             key = (order.company_id.id, order.partner_id.id, order.partner_shipping_id
380                 .id)
381             if key not in cache:
382                 cache[key] = self.env['account.fiscal.position'].with_company(
383                     order.company_id
384                 )._get_fiscal_position(order.partner_id, order.partner_shipping_id)
385             order.fiscal_position_id = cache[key]
386
387     @api.depends('partner_id')
388     def _compute_payment_term_id(self):
389         for order in self:
390             order = order.with_company(order.company_id)
391             order.payment_term_id = order.partner_id.property_payment_term_id
392
393     @api.depends('partner_id', 'company_id')
394     def _compute_pricelist_id(self):
395         for order in self:
396             if order.state != 'draft':
397                 continue
398             if not order.partner_id:
399                 order.pricelist_id = False
400                 continue
401             order = order.with_company(order.company_id)
402             order.pricelist_id = order.partner_id.property_product_pricelist
403
404     @api.depends('pricelist_id', 'company_id')
405     def _compute_currency_id(self):
406         for order in self:
407             order.currency_id = order.pricelist_id.currency_id or order.company_id.
408                 currency_id

```

```

404 @api.depends('currency_id', 'date_order', 'company_id')
405 def _compute_currency_rate(self):
406     for order in self:
407         order.currency_rate = self.env['res.currency']._get_conversion_rate(
408             from_currency=order.company_id.currency_id,
409             to_currency=order.currency_id,
410             company=order.company_id,
411             date=order.date_order.date(),
412         )
413
414 @api.depends('company_id')
415 def _compute_has_active_pricelist(self):
416     for order in self:
417         order.has_active_pricelist = bool(self.env['product.pricelist'].search(
418             [('company_id', 'in', (False, order.company_id.id)), ('active', '=',
419                 True)],
420             limit=1,
421         ))
422
423 @api.depends('partner_id')
424 def _compute_user_id(self):
425     for order in self:
426         if order.partner_id and not (order._origin.id and order.user_id):
427             # Recompute the salesman on partner change
428             # * if partner is set (is required anyway, so it will be set sooner
429             #   or later)
430             # * if the order is not saved or has no salesman already
431             order.user_id = (
432                 order.partner_id.user_id
433                 or order.partner_id.commercial_partner_id.user_id
434                 or (self.user_has_groups('sales_team.group_sale_salesman') and
435                     self.env.user)
436             )
437
438 @api.depends('partner_id', 'user_id')
439 def _compute_team_id(self):
440     cached_teams = {}
441     for order in self:
442         default_team_id = self.env.context.get('default_team_id', False) or order.
443         team_id.id or order.partner_id.team_id.id
444         user_id = order.user_id.id
445         company_id = order.company_id.id
446         key = (default_team_id, user_id, company_id)
447         if key not in cached_teams:
448             cached_teams[key] = self.env['crm.team'].with_context(
449                 default_team_id=default_team_id,
450                 user_id=user_id,
451                 domain=self.env['crm.team']._check_company_domain(company_id),
452             )
453         order.team_id = cached_teams[key]
454
455 @api.depends('order_line.price_subtotal', 'order_line.price_tax',
456     'order_line.price_total')
457 def _compute_amounts(self):
458     """Compute the total amounts of the SO."""
459     for order in self:
460         order_lines = order.order_line.filtered(lambda x: not x.display_type)
461
462         if order.company_id.tax_calculation_rounding_method == 'round_globally':
463             tax_results = self.env['account.tax']._compute_taxes([
464                 line._convert_to_tax_base_line_dict()
465                 for line in order_lines
466             ])
467             totals = tax_results['totals']
468             amount_untaxed = totals.get(order.currency_id, {}).get(
469                 'amount_untaxed', 0.0)
470             amount_tax = totals.get(order.currency_id, {}).get('amount_tax', 0.0)
471         else:
472             amount_untaxed = sum(order_lines.mapped('price_subtotal'))
473             amount_tax = sum(order_lines.mapped('price_tax'))

```



```

470         order.amount_untaxed = amount_untaxed
471         order.amount_tax = amount_tax
472         order.amount_total = order.amount_untaxed + order.amount_tax
473
474     @api.depends('order_line.invoice_lines')
475     def _get_invoiced(self):
476         # The invoice_ids are obtained thanks to the invoice lines of the SO
477         # lines, and we also search for possible refunds created directly from
478         # existing invoices. This is necessary since such a refund is not
479         # directly linked to the SO.
480         for order in self:
481             invoices = order.order_line.invoice_lines.move_id.filtered(lambda r: r.
482             move_type in ('out_invoice', 'out_refund'))
483             order.invoice_ids = invoices
484             order.invoice_count = len(invoices)
485
486     def _search_invoice_ids(self, operator, value):
487         if operator == 'in' and value:
488             self.env.cr.execute("""
489                 SELECT array_agg(so.id)
490                 FROM sale_order so
491                 JOIN sale_order_line sol ON sol.order_id = so.id
492                 JOIN sale_order_line_invoice_rel soli_rel ON
493                 soli_rel.order_line_id = sol.id
494                 JOIN account_move_line aml ON aml.id = soli_rel.invoice_line_id
495                 JOIN account_move am ON am.id = aml.move_id
496                 WHERE
497                     am.move_type in ('out_invoice', 'out_refund') AND
498                     am.id = ANY(%s)
499             """, (list(value),))
500             so_ids = self.env.cr.fetchone()[0] or []
501             return [('id', 'in', so_ids)]
502         elif operator == '=' and not value:
503             # special case for [('invoice_ids', '=', False)], i.e. "Invoices is not
504             # set"
505             # We cannot just search [('order_line.invoice_lines', '=', False)]
506             # because it returns orders with uninvoiced lines, which is not
507             # same "Invoices is not set" (some lines may have invoices and some
508             # doesn't)
509             # A solution is making inverted search first ("orders with invoiced
510             # lines") and then invert results ("get all other orders")
511             # Domain below returns subset of ('order_line.invoice_lines', '!=', False)
512             order_ids = self._search([
513                 ('order_line.invoice_lines.move_id.move_type', 'in', ('out_invoice',
514                 'out_refund'))
515             ])
516             return [('id', 'not in', order_ids)]
517         return [
518             ('order_line.invoice_lines.move_id.move_type', 'in', ('out_invoice',
519             'out_refund')),
520             ('order_line.invoice_lines.move_id', operator, value),
521         ]
522
523     @api.depends('state', 'order_line.invoice_status')
524     def _compute_invoice_status(self):
525         """
526         Compute the invoice status of a SO. Possible statuses:
527         - no: if the SO is not in status 'sale' or 'done', we consider that there is
528         nothing to
529         invoice. This is also the default value if the conditions of no other
530         status is met.
531         - to invoice: if any SO line is 'to invoice', the whole SO is 'to invoice'
532         - invoiced: if all SO lines are invoiced, the SO is invoiced.
533         - upselling: if all SO lines are invoiced or upselling, the status is
534         upselling.
535         """
536         confirmed_orders = self.filtered(lambda so: so.state == 'sale')
537         (self - confirmed_orders).invoice_status = 'no'
538         if not confirmed_orders:

```



```

534         return
535     line_invoice_status_all = [
536         (order.id, invoice_status)
537         for order, invoice_status in self.env['sale.order.line']._read_group([
538             ('order_id', 'in', confirmed_orders.ids),
539             ('is_downpayment', '=', False),
540             ('display_type', '=', False),
541         ],
542         ['order_id', 'invoice_status'])]
543     for order in confirmed_orders:
544         line_invoice_status = [d[1] for d in line_invoice_status_all if d[0] ==
545                                order.id]
546         if order.state != 'sale':
547             order.invoice_status = 'no'
548         elif any(invoice_status == 'to invoice' for invoice_status in
549                  line_invoice_status):
550             order.invoice_status = 'to invoice'
551         elif line_invoice_status and all(invoice_status == 'invoiced' for
552                                           invoice_status in line_invoice_status):
553             order.invoice_status = 'invoiced'
554         elif line_invoice_status and all(invoice_status in ('invoiced',
555                                                              'upselling') for invoice_status in
556                                           line_invoice_status):
557             order.invoice_status = 'upselling'
558         else:
559             order.invoice_status = 'no'
560
561     @api.depends('transaction_ids')
562     def _compute_authorized_transaction_ids(self):
563         for trans in self:
564             trans.authorized_transaction_ids = trans.transaction_ids.filtered(lambda t
565                                     : t.state == 'authorized')
566
567     @api.depends('transaction_ids')
568     def _compute_amount_paid(self):
569         """ Sum of the amount paid through all transactions for this SO. """
570         for order in self:
571             order.amount_paid = sum(
572                 tx.amount for tx in order.transaction_ids if tx.state in ('authorized'
573                                     , 'done')
574             )
575
576     def _compute_amount_undiscounted(self):
577         for order in self:
578             total = 0.0
579             for line in order.order_line:
580                 total += (line.price_subtotal * 100)/(100-line.discount) if line.
581                     discount != 100 else (line.price_unit * line.product_uom_qty)
582             order.amount_undiscounted = total
583
584     @api.depends('order_line.customer_lead', 'date_order', 'state')
585     def _compute_expected_date(self):
586         """ For service and consumable, we only take the min dates. This method is
587             extended in sale_stock to
588             take the picking_policy of SO into account.
589         """
590         self.mapped("order_line") # Prefetch indication
591         for order in self:
592             if order.state == 'cancel':
593                 order.expected_date = False
594                 continue
595             dates_list = order.order_line.filtered(
596                 lambda line: not line.display_type and not line._is_delivery()
597             ).mapped(lambda line: line and line._expected_date())
598             if dates_list:
599                 order.expected_date = min(dates_list)
600             else:
601                 order.expected_date = False
602
603     def _compute_is_expired(self):
604         today = fields.Date.today()
605         for order in self:
606             order.is_expired = order.state == 'sent' and order.validity_date and order

```

```

        .validity_date < today
598
599 @api.depends('company_id', 'fiscal_position_id')
600 def _compute_tax_country_id(self):
601     for record in self:
602         if record.fiscal_position_id.foreign_vat:
603             record.tax_country_id = record.fiscal_position_id.country_id
604         else:
605             record.tax_country_id = record.company_id.account_fiscal_country_id
606
607 @api.depends('invoice_ids.state', 'currency_id', 'amount_total')
608 def _compute_amount_to_invoice(self):
609     for order in self:
610         # If the invoice status is 'Fully Invoiced' force the amount to invoice
611         # to equal zero and return early.
612         if order.invoice_status == 'invoiced':
613             order.amount_to_invoice = 0.0
614             return
615
616         order.amount_to_invoice = order.amount_total
617         for invoice in order.invoice_ids.filtered(lambda x: x.state == 'posted'):
618             prices = sum(invoice.line_ids.filtered(lambda x: order in x.
619                 sale_line_ids.order_id).mapped('price_total'))
620             invoice_amount_currency = invoice.currency_id._convert(
621                 prices * -invoice.direction_sign,
622                 order.currency_id,
623                 invoice.company_id,
624                 invoice.date,
625             )
626             order.amount_to_invoice -= invoice_amount_currency
627
628 @api.depends('amount_total', 'amount_to_invoice')
629 def _compute_amount_invoiced(self):
630     for order in self:
631         order.amount_invoiced = order.amount_total - order.amount_to_invoice
632
633 @api.depends('company_id', 'partner_id', 'amount_total')
634 def _compute_partner_credit_warning(self):
635     for order in self:
636         order.with_company(order.company_id)
637         order.partner_credit_warning = ''
638         show_warning = order.state in ('draft', 'sent') and \
639             order.company_id.account_use_credit_limit
640         if show_warning:
641             order.partner_credit_warning = self.env['account.move'].
642             _build_credit_warning_message(
643                 order,
644                 current_amount=(order.amount_total / order.currency_rate),
645             )
646
647 @api.depends('order_line.tax_id', 'order_line.price_unit', 'amount_total',
648 'amount_untaxed', 'currency_id')
649 def _compute_tax_totals(self):
650     for order in self:
651         order_lines = order.order_line.filtered(lambda x: not x.display_type)
652         order.tax_totals = self.env['account.tax']._prepare_tax_totals(
653             [x._convert_to_tax_base_line_dict() for x in order_lines],
654             order.currency_id or order.company_id.currency_id,
655         )
656
657 @api.depends('state')
658 def _compute_type_name(self):
659     for record in self:
660         if record.state in ('draft', 'sent', 'cancel'):
661             record.type_name = _("Quotation")
662         else:
663             record.type_name = _("Sales Order")
664
665 # portal.mixin override
666 def _compute_access_url(self):
667     super()._compute_access_url()
668     for order in self:

```

```

665         order.access_url = f'/my/orders/{order.id}'
666
667     #=== CONSTRAINT METHODS ===#
668
669     @api.constrains('company_id', 'order_line')
670     def _check_order_line_company_id(self):
671         for order in self:
672             companies = order.order_line.product_id.company_id
673             if companies and companies != order.company_id:
674                 bad_products = order.order_line.product_id.filtered(lambda p: p.
675                     company_id and p.company_id != order.company_id)
676                 raise ValidationError(_(
677                     "Your quotation contains products from company
678                     %(product_company)s whereas your quotation belongs to company
679                     %(quote_company)s. \n Please change the company of your quotation
680                     or remove the products from other companies (%(bad_products)s).",
681                     product_company=', '.join(companies.mapped('display_name')),
682                     quote_company=order.company_id.display_name,
683                     bad_products=', '.join(bad_products.mapped('display_name')),
684                 ))
685
686     @api.constrains('prepayment_percent')
687     def _check_prepayment_percent(self):
688         for order in self:
689             if order.require_payment and not (0 < order.prepayment_percent <= 1.0):
690                 raise ValidationError(_("Prepayment percentage must be a valid
691                     percentage."))
692
693     #=== ONCHANGE METHODS ===#
694
695     @api.onchange('commitment_date', 'expected_date')
696     def _onchange_commitment_date(self):
697         """ Warn if the commitment dates is sooner than the expected date """
698         if self.commitment_date and self.expected_date and self.commitment_date < self
699             .expected_date:
700             return {
701                 'warning': {
702                     'title': _('Requested date is too soon.'),
703                     'message': _("The delivery date is sooner than the expected date."
704                         " You may be unable to honor the delivery date.")
705                 }
706             }
707
708     @api.onchange('company_id')
709     def _onchange_company_id_warning(self):
710         self.show_update_pricelist = True
711         if self.order_line and self.state == 'draft':
712             return {
713                 'warning': {
714                     'title': _("Warning for the change of your quotation's company"),
715                     'message': _("Changing the company of an existing quotation might
716                         need some "
717                             "manual adjustments in the details of the lines. You
718                             might "
719                             "consider updating the prices."),
720                 }
721             }
722
723     @api.onchange('fiscal_position_id')
724     def _onchange_fpos_id_show_update_fpos(self):
725         if self.order_line and (
726             not self.fiscal_position_id
727             or (self.fiscal_position_id and self._origin.fiscal_position_id != self.
728                 fiscal_position_id)
729         ):
730             self.show_update_fpos = True
731
732     @api.onchange('partner_id')
733     def _onchange_partner_id_warning(self):
734         if not self.partner_id:
735             return

```

```

728         partner = self.partner_id
729
730         # If partner has no warning, check its company
731         if partner.sale_warn == 'no-message' and partner.parent_id:
732             partner = partner.parent_id
733
734         if partner.sale_warn and partner.sale_warn != 'no-message':
735             # Block if partner only has warning but parent company is blocked
736             if partner.sale_warn != 'block' and partner.parent_id and partner.
737                 parent_id.sale_warn == 'block':
738                 partner = partner.parent_id
739
740             if partner.sale_warn == 'block':
741                 self.partner_id = False
742
743             return {
744                 'warning': {
745                     'title': _("Warning for %s", partner.name),
746                     'message': partner.sale_warn_msg,
747                 }
748             }
749
750         @api.onchange('pricelist_id')
751         def _onchange_pricelist_id_show_update_prices(self):
752             self.show_update_pricelist = bool(self.order_line)
753
754         @api.onchange('prepayment_percent')
755         def _onchange_prepayment_percent(self):
756             if not self.prepayment_percent:
757                 self.require_payment = False
758
759         #=== CRUD METHODS ===#
760
761         @api.model_create_multi
762         def create(self, vals_list):
763             for vals in vals_list:
764                 if 'company_id' in vals:
765                     self = self.with_company(vals['company_id'])
766                 if vals.get('name', _("New")) == _("New"):
767                     seq_date = fields.Datetime.context_timestamp(
768                         self, fields.Datetime.to_datetime(vals['date_order']))
769                     if 'date_order' in vals else None
770                     vals['name'] = self.env['ir.sequence'].next_by_code(
771                         'sale.order', sequence_date=seq_date) or _("New")
772
773             return super().create(vals_list)
774
775         def copy_data(self, default=None):
776             if default is None:
777                 default = {}
778             if 'order_line' not in default:
779                 default['order_line'] = [
780                     Command.create(line.copy_data()[0])
781                     for line in self.order_line.filtered(lambda l: not l.is_downpayment)
782                 ]
783             return super().copy_data(default)
784
785         @api.ondelete(at_uninstall=False)
786         def _unlink_except_draft_or_cancel(self):
787             for order in self:
788                 if order.state not in ('draft', 'cancel'):
789                     raise UserError(_(
790                         "You can not delete a sent quotation or a confirmed sales order."
791                         " You must first cancel it.))
792
793         #=== ACTION METHODS ===#
794
795         def action_open_discount_wizard(self):
796             self.ensure_one()
797             return {
798                 'name': _("Discount"),
799                 'type': 'ir.actions.act_window',

```

```

799         'res_model': 'sale.order.discount',
800         'view_mode': 'form',
801         'target': 'new',
802     }
803
804     def action_draft(self):
805         orders = self.filtered(lambda s: s.state in ['cancel', 'sent'])
806         return orders.write({
807             'state': 'draft',
808             'signature': False,
809             'signed_by': False,
810             'signed_on': False,
811         })
812
813     def action_quotation_send(self):
814         """ Opens a wizard to compose an email, with relevant mail template loaded by
            default """
815         self.ensure_one()
816         self.order_line.validate_analytic_distribution()
817         lang = self.env.context.get('lang')
818         mail_template = self._find_mail_template()
819         if mail_template and mail_template.lang:
820             lang = mail_template._render_lang(self.ids)[self.id]
821         ctx = {
822             'default_model': 'sale.order',
823             'default_res_ids': self.ids,
824             'default_template_id': mail_template.id if mail_template else None,
825             'default_composition_mode': 'comment',
826             'mark_so_as_sent': True,
827             'default_email_layout_xmlid':
828                 'mail.mail_notification_layout_with_responsible_signature',
829             'proforma': self.env.context.get('proforma', False),
830             'force_email': True,
831             'model_description': self.with_context(lang=lang).type_name,
832         }
833         return {
834             'type': 'ir.actions.act_window',
835             'view_mode': 'form',
836             'res_model': 'mail.compose.message',
837             'views': [(False, 'form')],
838             'view_id': False,
839             'target': 'new',
840             'context': ctx,
841         }
842
843     def _find_mail_template(self):
844         """ Get the appropriate mail template for the current sales order based on
            its state.
845
846         If the SO is confirmed, we return the mail template for the sale confirmation.
847         Otherwise, we return the quotation email template.
848
849         :return: The correct mail template based on the current status
850         :rtype: record of `mail.template` or `None` if not found
851         """
852         self.ensure_one()
853         if self.env.context.get('proforma') or self.state != 'sale':
854             return self.env.ref('sale.email_template_edi_sale', raise_if_not_found=False)
855         else:
856             return self._get_confirmation_template()
857
858     def _get_confirmation_template(self):
859         """ Get the mail template sent on SO confirmation (or for confirmed SO's).
860
861         :return: `mail.template` record or None if default template wasn't found
862         """
863         self.ensure_one()
864         default_confirmation_template_id = self.env['ir.config_parameter'].sudo().get_param(
865             'sale.default_confirmation_template'
866         )

```

```

866         default_confirmation_template = default_confirmation_template_id \
867             and self.env['mail.template'].browse(int(default_confirmation_template_id)
868             ).exists()
869         if default_confirmation_template:
870             return default_confirmation_template
871         else:
872             return self.env.ref('sale.mail_template_sale_confirmation',
873                                 raise_if_not_found=False)
874
875     def action_quotation_sent(self):
876         """ Mark the given draft quotation(s) as sent.
877
878         :raise: UserError if any given SO is not in draft state.
879         """
880         if any(order.state != 'draft' for order in self):
881             raise UserError(_("Only draft orders can be marked as sent directly."))
882
883         for order in self:
884             order.message_subscribe(partner_ids=order.partner_id.ids)
885
886         self.write({'state': 'sent'})
887
888     def action_confirm(self):
889         """ Confirm the given quotation(s) and set their confirmation date.
890
891         If the corresponding setting is enabled, also locks the Sale Order.
892
893         :return: True
894         :rtype: bool
895         :raise: UserError if trying to confirm cancelled SO's
896         """
897         if not all(order._can_be_confirmed() for order in self):
898             raise UserError(_("The following orders are not in a state requiring confirmation: %s",
899                               ", ".join(self.mapped('display_name')),
900                               ))
901
902         self.order_line._validate_analytic_distribution()
903
904         for order in self:
905             order.validate_taxes_on_sales_order()
906             if order.partner_id in order.message_partner_ids:
907                 continue
908             order.message_subscribe([order.partner_id.id])
909
910         self.write(self._prepare_confirmation_values())
911
912         # Context key 'default_name' is sometimes propagated up to here.
913         # We don't need it and it creates issues in the creation of linked records.
914         context = self._context.copy()
915         context.pop('default_name', None)
916
917         self.with_context(context)._action_confirm()
918         if self.env.user.has_group('sale.group_auto_done_setting'):
919             self.action_lock()
920
921         return True
922
923     def _can_be_confirmed(self):
924         self.ensure_one()
925         return self.state in {'draft', 'sent'}
926
927     def _prepare_confirmation_values(self):
928         """ Prepare the sales order confirmation values.
929
930         Note: self can contain multiple records.
931
932         :return: Sales Order confirmation values
933         :rtype: dict
934         """
935         return {
936             'state': 'sale',

```

```

936         'date_order': fields.Datetime.now()
937     }
938
939     def _action_confirm(self):
940         """ Implementation of additional mechanism of Sales Order confirmation.
941             This method should be extended when the confirmation should generated
942             other documents. In this method, the SO are in 'sale' state (not yet
943             'done').
944         """
945         # create an analytic account if at least an expense product
946         for order in self:
947             if any(expense_policy not in [False, 'no'] for expense_policy in order.
948                 order_line.product_id.mapped('expense_policy')):
949                 if not order.analytic_account_id:
950                     order._create_analytic_account()
951
952     def _send_order_confirmation_mail(self):
953         """ Send a mail to the SO customer to inform them that their order has been
954             confirmed.
955
956             :return: None
957         """
958         for order in self:
959             mail_template = order._get_confirmation_template()
960             order._send_order_notification_mail(mail_template)
961
962     def _send_payment_succeeded_for_order_mail(self):
963         """ Send a mail to the SO customer to inform them that a payment has been
964             initiated.
965
966             :return: None
967         """
968         mail_template = self.env.ref(
969             'sale.mail_template_sale_payment_executed', raise_if_not_found=False
970         )
971         for order in self:
972             order._send_order_notification_mail(mail_template)
973
974     def _send_order_notification_mail(self, mail_template):
975         """ Send a mail to the customer
976
977             Note: self.ensure_one()
978
979             :param mail.template mail_template: the template used to generate the mail
980             :return: None
981         """
982         self.ensure_one()
983
984         if not mail_template:
985             return
986
987         if self.env.su:
988             # sending mail in sudo was meant for it being sent from superuser
989             self = self.with_user(SUPERUSER_ID)
990
991         self.with_context(force_send=True).message_post_with_source(
992             mail_template,
993             email_layout_xmlid=
994                 'mail.mail_notification_layout_with_responsible_signature',
995             subtype_xmlid='mail.mt_comment',
996         )
997
998     def action_lock(self):
999         for order in self:
1000             tx = order.sudo().transaction_ids.get_last()
1001             if tx and tx.state == 'pending' and tx.provider_id.code == 'custom' and tx
1002                 .provider_id.custom_mode == 'wire_transfer':
1003                 tx._set_done()
1004                 tx.write({'is_post_processed': True})
1005             self.locked = True
1006
1007     def action_unlock(self):

```



```

1002         self.locked = False
1003
1004     def action_cancel(self):
1005         """ Cancel SO after showing the cancel wizard when needed. (cfr
1006             :meth: `_show_cancel_wizard`)
1007
1008         For post-cancel operations, please only override :meth: `_action_cancel`.
1009
1010         note: self.ensure_one() if the wizard is shown.
1011         """
1012         if any(order.locked for order in self):
1013             raise UserError(_("You cannot cancel a locked order. Please unlock it
1014                 first."))
1015         cancel_warning = self._show_cancel_wizard()
1016         if cancel_warning:
1017             self.ensure_one()
1018             template_id = self.env['ir.model.data']._xmlid_to_res_id(
1019                 'sale.mail_template_sale_cancellation', raise_if_not_found=False
1020             )
1021             lang = self.env.context.get('lang')
1022             template = self.env['mail.template'].browse(template_id)
1023             if template.lang:
1024                 lang = template._render_lang(self.ids)[self.id]
1025             ctx = {
1026                 'default_template_id': template_id,
1027                 'default_order_id': self.id,
1028                 'mark_so_as_canceled': True,
1029                 'default_email_layout_xmlid':
1030                     "mail.mail_notification_layout_with_responsible_signature",
1031                 'model_description': self.with_context(lang=lang).type_name,
1032             }
1033             return {
1034                 'name': _('Cancel %s', self.type_name),
1035                 'view_mode': 'form',
1036                 'res_model': 'sale.order.cancel',
1037                 'view_id': self.env.ref('sale.sale_order_cancel_view_form').id,
1038                 'type': 'ir.actions.act_window',
1039                 'context': ctx,
1040                 'target': 'new'
1041             }
1042         else:
1043             return self._action_cancel()
1044
1045     def _action_cancel(self):
1046         inv = self.invoice_ids.filtered(lambda inv: inv.state == 'draft')
1047         inv.button_cancel()
1048         return self.write({'state': 'cancel'})
1049
1050     def _show_cancel_wizard(self):
1051         """ Decide whether the sale.order.cancel wizard should be shown to cancel
1052             specified orders.
1053
1054             :return: True if there is any non-draft order in the given orders
1055             :rtype: bool
1056         """
1057         if self.env.context.get('disable_cancel_warning'):
1058             return False
1059         return any(so.state != 'draft' for so in self)
1060
1061     def action_preview_sale_order(self):
1062         self.ensure_one()
1063         return {
1064             'type': 'ir.actions.act_url',
1065             'target': 'self',
1066             'url': self.get_portal_url(),
1067         }
1068
1069     def action_update_taxes(self):
1070         self.ensure_one()
1071
1072         self._recompute_taxes()
1073

```

```

1070         if self.partner_id:
1071             self.message_post(body=_("Product taxes have been recomputed according to
1072                                     fiscal position %s.",
1073                                     self.fiscal_position_id._get_html_link() if self.fiscal_position_id
1074                                     else ""))
1075
1076     def _recompute_taxes(self):
1077         lines_to_recompute = self.order_line.filtered(lambda line: not line.
1078                                                         display_type)
1079         lines_to_recompute._compute_tax_id()
1080         self.show_update_fpos = False
1081
1082     def action_update_prices(self):
1083         self.ensure_one()
1084
1085         self._recompute_prices()
1086
1087         if self.pricelist_id:
1088             message = _("Product prices have been recomputed according to pricelist
1089                         %s.",
1090                         self.pricelist_id._get_html_link())
1091         else:
1092             message = _("Product prices have been recomputed.")
1093         self.message_post(body=message)
1094
1095     def _recompute_prices(self):
1096         lines_to_recompute = self._get_update_prices_lines()
1097         lines_to_recompute.invalidate_recordset(['pricelist_item_id'])
1098         lines_to_recompute._compute_price_unit()
1099         # Special case: we want to overwrite the existing discount on
1100         # _recompute_prices call
1101         # i.e. to make sure the discount is correctly reset
1102         # if pricelist discount_policy is different than when the price was first
1103         # computed.
1104         lines_to_recompute.discount = 0.0
1105         lines_to_recompute._compute_discount()
1106         self.show_update_pricelist = False
1107
1108     def _default_order_line_values(self):
1109         default_data = super()._default_order_line_values()
1110         new_default_data = self.env['sale.order.line']._get_product_catalog_lines_data
1111         ()
1112         return {**default_data, **new_default_data}
1113
1114     def _get_action_add_from_catalog_extra_context(self):
1115         return {
1116             **super()._get_action_add_from_catalog_extra_context(),
1117             'product_catalog_currency_id': self.currency_id.id,
1118             'product_catalog_digits': self.order_line._fields['price_unit'].get_digits
1119             (self.env),
1120         }
1121
1122     def _get_product_catalog_domain(self):
1123         return expression.AND([super()._get_product_catalog_domain(), [('sale_ok', '=',
1124                                     , True)]]))
1125
1126     # INVOICING #
1127
1128     def _prepare_invoice(self):
1129         """
1130         Prepare the dict of values to create the new invoice for a sales order. This
1131         method may be
1132         overridden to implement custom invoice generation (making sure to call
1133         super() to establish
1134         a clean extension chain).
1135         """
1136         self.ensure_one()
1137
1138         values = {
1139             'ref': self.client_order_ref or '',
1140             'move_type': 'out_invoice',

```

```

1131         'narration': self.note,
1132         'currency_id': self.currency_id.id,
1133         'campaign_id': self.campaign_id.id,
1134         'medium_id': self.medium_id.id,
1135         'source_id': self.source_id.id,
1136         'team_id': self.team_id.id,
1137         'partner_id': self.partner_invoice_id.id,
1138         'partner_shipping_id': self.partner_shipping_id.id,
1139         'fiscal_position_id': (self.fiscal_position_id or self.fiscal_position_id.
_get_fiscal_position(self.partner_invoice_id)).id,
1140         'invoice_origin': self.name,
1141         'invoice_payment_term_id': self.payment_term_id.id,
1142         'invoice_user_id': self.user_id.id,
1143         'payment_reference': self.reference,
1144         'transaction_ids': [Command.set(self.transaction_ids.ids)],
1145         'company_id': self.company_id.id,
1146         'invoice_line_ids': [],
1147         'user_id': self.user_id.id,
1148     }
1149     if self.journal_id:
1150         values['journal_id'] = self.journal_id.id
1151     return values
1152
1153 def action_view_invoice(self, invoices=False):
1154     if not invoices:
1155         invoices = self.mapped('invoice_ids')
1156     action = self.env['ir.actions.actions']._for_xml_id(
        'account.action_move_out_invoice_type')
1157     if len(invoices) > 1:
1158         action['domain'] = [('id', 'in', invoices.ids)]
1159     elif len(invoices) == 1:
1160         form_view = [(self.env.ref('account.view_move_form').id, 'form')]
1161         if 'views' in action:
1162             action['views'] = form_view + [(state, view) for state, view in action[
                'views'] if view != 'form']
1163         else:
1164             action['views'] = form_view
1165         action['res_id'] = invoices.id
1166     else:
1167         action = {'type': 'ir.actions.act_window_close'}
1168
1169     context = {
1170         'default_move_type': 'out_invoice',
1171     }
1172     if len(self) == 1:
1173         context.update({
1174             'default_partner_id': self.partner_id.id,
1175             'default_partner_shipping_id': self.partner_shipping_id.id,
1176             'default_invoice_payment_term_id': self.payment_term_id.id or self.
partner_id.property_payment_term_id.id or self.env['account.move'].
default_get(['invoice_payment_term_id']).get('invoice_payment_term_id'
        ),
1177             'default_invoice_origin': self.name,
1178         })
1179     action['context'] = context
1180     return action
1181
1182 def _get_invoice_grouping_keys(self):
1183     return ['company_id', 'partner_id', 'currency_id']
1184
1185 def _nothing_to_invoice_error_message(self):
1186     return (
1187         "Cannot create an invoice. No items are available to invoice.\n\n"
1188         "To resolve this issue, please ensure that:\n"
1189         "    \u2022 The products have been delivered before attempting to invoice
them.\n"
1190         "    \u2022 The invoicing policy of the product is configured
correctly.\n\n"
1191         "If you want to invoice based on ordered quantities instead:\n"
1192         "    \u2022 For consumable or storable products, open the product, go to
the 'General Information' tab and change the 'Invoicing Policy' from
'Delivered Quantities' to 'Ordered Quantities'.\n"

```

```

1193         "\u2022 For services (and other products), change the 'Invoicing
        Policy' to 'Prepaid/Fixed Price'.\n"
1194     )
1195
1196     def _get_update_prices_lines(self):
1197         """ Hook to exclude specific lines which should not be updated based on price
        list recomputation """
1198         return self.order_line.filtered(lambda line: not line.display_type)
1199
1200     def _get_invoiceable_lines(self, final=False):
1201         """Return the invoiceable lines for order `self`."""
1202         down_payment_line_ids = []
1203         invoiceable_line_ids = []
1204         pending_section = None
1205         precision = self.env['decimal.precision'].precision_get('Product Unit of
        Measure')
1206
1207         for line in self.order_line:
1208             if line.display_type == 'line_section':
1209                 # Only invoice the section if one of its lines is invoiceable
1210                 pending_section = line
1211                 continue
1212             if line.display_type != 'line_note' and float_is_zero(line.qty_to_invoice,
        precision_digits=precision):
1213                 continue
1214             if line.qty_to_invoice > 0 or (line.qty_to_invoice < 0 and final) or line.
        display_type == 'line_note':
1215                 if line.is_downpayment:
1216                     # Keep down payment lines separately, to put them together
1217                     # at the end of the invoice, in a specific dedicated section.
1218                     down_payment_line_ids.append(line.id)
1219                     continue
1220                 if pending_section:
1221                     invoiceable_line_ids.append(pending_section.id)
1222                     pending_section = None
1223                     invoiceable_line_ids.append(line.id)
1224
1225         return self.env['sale.order.line'].browse(invoiceable_line_ids +
        down_payment_line_ids)
1226
1227     def _create_invoices(self, grouped=False, final=False, date=None):
1228         """ Create invoice(s) for the given Sales Order(s).
1229
1230         :param bool grouped: if True, invoices are grouped by SO id.
1231             If False, invoices are grouped by keys returned by
1232             :meth:`_get_invoice_grouping_keys`
1233         :param bool final: if True, refunds will be generated if necessary
1234         :param date: unused parameter
1235         :returns: created invoices
1236         :rtype: `account.move` recordset
1237         :raises: UserError if one of the orders has no invoiceable lines.
1238         """
1238         if not self.env['account.move'].check_access_rights('create', False):
1239             try:
1240                 self.check_access_rights('write')
1241                 self.check_access_rule('write')
1242             except AccessError:
1243                 return self.env['account.move']
1244
1245         # 1) Create invoices.
1246         invoice_vals_list = []
1247         invoice_item_sequence = 0 # Incremental sequencing to keep the lines order on
        the invoice.
1248         for order in self:
1249             order = order.with_company(order.company_id).with_context(lang=order.
        partner_invoice_id.lang)
1250
1251             invoice_vals = order._prepare_invoice()
1252             invoiceable_lines = order._get_invoiceable_lines(final)
1253
1254             if not any(not line.display_type for line in invoiceable_lines):
1255                 continue

```

```

1256 invoice_line_vals = []
1257 down_payment_section_added = False
1258 for line in invoiceable_lines:
1259     if not down_payment_section_added and line.is_downpayment:
1260         # Create a dedicated section for the down payments
1261         # (put at the end of the invoiceable_lines)
1262         invoice_line_vals.append(
1263             Command.create(
1264                 order._prepare_down_payment_section_line(sequence=
1265                     invoice_item_sequence)
1266             ),
1267         )
1268         down_payment_section_added = True
1269         invoice_item_sequence += 1
1270 invoice_line_vals.append(
1271     Command.create(
1272         line._prepare_invoice_line(sequence=invoice_item_sequence)
1273     ),
1274 )
1275 invoice_item_sequence += 1
1276
1277 invoice_vals['invoice_line_ids'] += invoice_line_vals
1278 invoice_vals_list.append(invoice_vals)
1279
1280 if not invoice_vals_list and self._context.get('raise_if_nothing_to_invoice',
1281 True):
1282     raise UserError(self._nothing_to_invoice_error_message())
1283
1284 # 2) Manage 'grouped' parameter: group by (partner_id, currency_id).
1285 if not grouped:
1286     new_invoice_vals_list = []
1287     invoice_grouping_keys = self._get_invoice_grouping_keys()
1288     invoice_vals_list = sorted(
1289         invoice_vals_list,
1290         key=lambda x: [
1291             x.get(grouping_key) for grouping_key in invoice_grouping_keys
1292         ]
1293     )
1294     for _grouping_keys, invoices in groupby(invoice_vals_list, key=lambda x: [
1295         x.get(grouping_key) for grouping_key in invoice_grouping_keys]):
1296         origins = set()
1297         payment_refs = set()
1298         refs = set()
1299         ref_invoice_vals = None
1300         for invoice_vals in invoices:
1301             if not ref_invoice_vals:
1302                 ref_invoice_vals = invoice_vals
1303             else:
1304                 ref_invoice_vals['invoice_line_ids'] += invoice_vals[
1305                     'invoice_line_ids']
1306                 origins.add(invoice_vals['invoice_origin'])
1307                 payment_refs.add(invoice_vals['payment_reference'])
1308                 refs.add(invoice_vals['ref'])
1309             ref_invoice_vals.update({
1310                 'ref': ', '.join(refs)[:2000],
1311                 'invoice_origin': ', '.join(origins),
1312                 'payment_reference': len(payment_refs) == 1 and payment_refs.pop()
1313                 or False,
1314             })
1315             new_invoice_vals_list.append(ref_invoice_vals)
1316 invoice_vals_list = new_invoice_vals_list
1317
1318 # 3) Create invoices.
1319
1320 # As part of the invoice creation, we make sure the sequence of multiple SO
1321 do not interfere
1322 # in a single invoice. Example:
1323 # SO 1:
1324 # - Section A (sequence: 10)
1325 # - Product A (sequence: 11)
1326 # SO 2:

```

```

1322 # - Section B (sequence: 10)
1323 # - Product B (sequence: 11)
1324 #
1325 # If SO 1 & 2 are grouped in the same invoice, the result will be:
1326 # - Section A (sequence: 10)
1327 # - Section B (sequence: 10)
1328 # - Product A (sequence: 11)
1329 # - Product B (sequence: 11)
1330 #
1331 # Resequencing should be safe, however we resequence only if there are less
1332 # invoices than
1333 # orders, meaning a grouping might have been done. This could also mean that
1334 # only a part
1335 # of the selected SO are invoiceable, but resequencing in this case shouldn't
1336 # be an issue.
1337 if len(invoice_vals_list) < len(self):
1338     SaleOrderLine = self.env['sale.order.line']
1339     for invoice in invoice_vals_list:
1340         sequence = 1
1341         for line in invoice['invoice_line_ids']:
1342             line[2]['sequence'] = SaleOrderLine._get_invoice_line_sequence(new
1343                                     =sequence, old=line[2]['sequence'])
1344             sequence += 1
1345
1346 # Manage the creation of invoices in sudo because a salesperson must be able
1347 # to generate an invoice from a
1348 # sale order without "billing" access rights. However, he should not be able
1349 # to create an invoice from scratch.
1350 moves = self.env['account.move'].sudo().with_context(default_move_type=
1351 'out_invoice').create(invoice_vals_list)
1352
1353 # 4) Some moves might actually be refunds: convert them if the total amount
1354 # is negative
1355 # We do this after the moves have been created since we need taxes, etc. to
1356 # know if the total
1357 # is actually negative or not
1358 if final:
1359     moves.sudo().filtered(lambda m: m.amount_total < 0).
1360     action_switch_move_type()
1361 for move in moves:
1362     if final:
1363         # Downpayment might have been determined by a fixed amount set by the
1364         # user.
1365         # This amount is tax included. This can lead to rounding issues.
1366         # E.g. a user wants a 100€ DP on a product with 21% tax.
1367         # 100 / 1.21 = 82.64, 82.64 * 1,21 = 99.99
1368         # This is already corrected by adding/removing the missing cents on
1369         # the DP invoice,
1370         # but must also be accounted for on the final invoice.
1371
1372         delta_amount = 0
1373         for order_line in self.order_line:
1374             if not order_line.is_downpayment:
1375                 continue
1376             inv_amt = order_amt = 0
1377             for invoice_line in order_line.invoice_lines:
1378                 if invoice_line.move_id == move:
1379                     inv_amt += invoice_line.price_total
1380                 elif invoice_line.move_id.state != 'cancel': # filter out
1381                     canceled dp lines
1382                     order_amt += invoice_line.price_total
1383             if inv_amt and order_amt:
1384                 # if not inv_amt, this order line is not related to current
1385                 # move
1386                 # if no order_amt, dp order line was not invoiced
1387                 delta_amount += (inv_amt * (1 if move.is_inbound() else -1)) +
1388                     order_amt
1389
1390         if not move.currency_id.is_zero(delta_amount):
1391             receivable_line = move.line_ids.filtered(
1392                 lambda aml: aml.account_id.account_type == 'asset_receivable'
1393             )[:1]

```

```

1378         product_lines = move.line_ids.filtered(
1379             lambda aml: aml.display_type == 'product' and aml.
1380                 is_downpayment)
1381         tax_lines = move.line_ids.filtered(
1382             lambda aml: aml.tax_line_id.amount_type not in (False, 'fixed'
1383             ))
1384         if tax_lines and product_lines and receivable_line:
1385             line_commands = [Command.update(receivable_line.id, {
1386                 'amount_currency': receivable_line.amount_currency +
1387                 delta_amount,
1388             })]
1389             delta_sign = 1 if delta_amount > 0 else -1
1390             for lines, attr, sign in (
1391                 (product_lines, 'price_total', -1 if move.is_inbound()
1392                 else 1),
1393                 (tax_lines, 'amount_currency', 1),
1394             ):
1395                 remaining = delta_amount
1396                 lines_len = len(lines)
1397                 for line in lines:
1398                     if move.currency_id.compare_amounts(remaining, 0) !=
1399                         delta_sign:
1400                         break
1401                     amt = delta_sign * max(
1402                         move.currency_id.rounding,
1403                         abs(move.currency_id.round(remaining / lines_len
1404                         ))),
1405                     )
1406                     remaining -= amt
1407                     line_commands.append(Command.update(line.id, {attr:
1408                         line[attr] + amt * sign}))
1409                 move.line_ids = line_commands
1410
1411         move.message_post_with_source(
1412             'mail.message_origin_link',
1413             render_values={'self': move, 'origin': move.line_ids.sale_line_ids.
1414                 order_id},
1415             subtype_xmlid='mail.mt_note',
1416         )
1417     return moves
1418
1419 # MAIL #
1420
1421 def _track_finalize(self):
1422     """ Override of `mail` to prevent logging changes when the SO is in a draft
1423     state. """
1424     if (len(self) == 1
1425         # The method _track_finalize is sometimes called too early or too late
1426         # and it
1427         # might cause a desynchronization with the cache, thus this condition is
1428         # needed.
1429         and self.env.cache.contains(self, self._fields['state']) and self.state ==
1430             'draft'):
1431         self.env.cr.precommit.data.pop(f'mail.tracking.{self._name}', {})
1432         self.env.flush_all()
1433         return
1434     return super()._track_finalize()
1435
1436 @api.returns('mail.message', lambda value: value.id)
1437 def message_post(self, **kwargs):
1438     if self.env.context.get('mark_so_as_sent'):
1439         self.filtered(lambda o: o.state == 'draft').with_context(tracking_disable=
1440             True).write({'state': 'sent'})
1441     so_ctx = {'mail_post_autofollow': self.env.context.get('mail_post_autofollow',
1442         True)}
1443     if self.env.context.get('mark_so_as_sent') and 'mail_notify_author' not in
1444         kwargs:
1445         kwargs['notify_author'] = self.env.user.partner_id.id in (kwargs.get(
1446             'partner_ids') or [])
1447     return super(SaleOrder, self.with_context(**so_ctx)).message_post(**kwargs)
1448
1449 def _notify_get_recipients_groups(self, message, model_description, msg_vals=None

```



```

):
1434     """ Give access button to users and portal customer as portal is integrated
1435     in sale. Customer and portal group have probably no right to see
1436     the document so they don't have the access button. """
1437     groups = super()._notify_get_recipients_groups(
1438         message, model_description, msg_vals=msg_vals
1439     )
1440     if not self:
1441         return groups
1442
1443     self.ensure_one()
1444     if self._context.get('proforma'):
1445         for group in [g for g in groups if g[0] in ('portal_customer', 'portal',
1446             'follower', 'customer')]:
1447             group[2]['has_button_access'] = False
1448         return groups
1449     local_msg_vals = dict(msg_vals or {})
1450
1451     # portal customers have full access (existence not granted, depending on
1452     partner_id)
1453     try:
1454         customer_portal_group = next(group for group in groups if group[0] ==
1455             'portal_customer')
1456     except StopIteration:
1457         pass
1458     else:
1459         access_opt = customer_portal_group[2].setdefault('button_access', {})
1460         is_tx_pending = self.get_portal_last_transaction().state == 'pending'
1461         if self._has_to_be_signed():
1462             if self._has_to_be_paid():
1463                 access_opt['title'] = _("View Quotation") if is_tx_pending else _(
1464                     "Sign & Pay Quotation")
1465             else:
1466                 access_opt['title'] = _("Accept & Sign Quotation")
1467         elif self._has_to_be_paid() and not is_tx_pending:
1468             access_opt['title'] = _("Accept & Pay Quotation")
1469         elif self.state in ('draft', 'sent'):
1470             access_opt['title'] = _("View Quotation")
1471
1472     # enable followers that have access through portal
1473     follower_group = next(group for group in groups if group[0] == 'follower')
1474     follower_group[2]['active'] = True
1475     follower_group[2]['has_button_access'] = True
1476     access_opt = follower_group[2].setdefault('button_access', {})
1477     if self.state in ('draft', 'sent'):
1478         access_opt['title'] = _("View Quotation")
1479     else:
1480         access_opt['title'] = _("View Order")
1481     access_opt['url'] = self._notify_get_action_link('view', **local_msg_vals)
1482
1483     return groups
1484
1485 def _notify_by_email_prepare_rendering_context(self, message, msg_vals,
1486     model_description=False,
1487
1488         force_email_company=False,
1489         force_email_lang=False):
1490     render_context = super()._notify_by_email_prepare_rendering_context(
1491         message, msg_vals, model_description=model_description,
1492         force_email_company=force_email_company, force_email_lang=force_email_lang
1493     )
1494     lang_code = render_context.get('lang')
1495     subtitles = [
1496         render_context['record'].name,
1497     ]
1498
1499     if self.amount_total:
1500         # Do not show the price in subtitles if zero (e.g. e-commerce orders are
1501         created empty)
1502         subtitles.append(
1503             format_amount(self.env, self.amount_total, self.currency_id, lang_code
1504                 =lang_code),
1505         )

```

```

1497
1498     if self.validity_date and self.state in ['draft', 'sent']:
1499         formatted_date = format_date(self.env, self.validity_date, lang_code=
            lang_code)
1500         subtitles.append(_("Expires on %(date)s", date=formatted_date))
1501
1502     render_context['subtitles'] = subtitles
1503     return render_context
1504
1505 def _phone_get_number_fields(self):
1506     """ No phone or mobile field is available on sale model. Instead SMS will
1507     fallback on partner-based computation using ``_mail_get_partner_fields``. """
1508     return []
1509
1510 def _track_subtype(self, init_values):
1511     self.ensure_one()
1512     if 'state' in init_values and self.state == 'sale':
1513         return self.env.ref('sale.mt_order_confirmed')
1514     elif 'state' in init_values and self.state == 'sent':
1515         return self.env.ref('sale.mt_order_sent')
1516     return super()._track_subtype(init_values)
1517
1518 # PAYMENT #
1519
1520 def _force_lines_to_invoice_policy_order(self):
1521     """Force the qty_to_invoice to be computed as if the invoice_policy
1522     was set to "Ordered quantities", independently of the product configuration.
1523
1524     This is needed for the automatic invoice logic, as we want to automatically
1525     invoice the full SO when it's paid.
1526     """
1527     for line in self.order_line:
1528         if line.state == 'sale':
1529             # No need to set 0 as it is already the standard logic in the compute
1530             # method.
1531             line.qty_to_invoice = line.product_uom_qty - line.qty_invoiced
1532
1533 def payment_action_capture(self):
1534     """ Capture all transactions linked to this sale order. """
1535     self.ensure_one()
1536     payment_utils.check_rights_on_recordset(self)
1537
1538     # In sudo mode to bypass the checks on the rights on the transactions.
1539     return self.transaction_ids.sudo().action_capture()
1540
1541 def payment_action_void(self):
1542     """ Void all transactions linked to this sale order. """
1543     payment_utils.check_rights_on_recordset(self)
1544
1545     # In sudo mode to bypass the checks on the rights on the transactions.
1546     self.authorized_transaction_ids.sudo().action_void()
1547
1548 def get_portal_last_transaction(self):
1549     self.ensure_one()
1550     return self.transaction_ids.sudo()._get_last()
1551
1552 def _get_order_lines_to_report(self):
1553     down_payment_lines = self.order_line.filtered(lambda line:
1554         line.is_downpayment
1555         and not line.display_type
1556         and not line._get_downpayment_state()
1557     )
1558
1559 def show_line(line):
1560     if not line.is_downpayment:
1561         return True
1562     elif line.display_type and down_payment_lines:
1563         return True # Only show the down payment section if down payments
1564                     # were posted
1565     elif line in down_payment_lines:
1566         return True # Only show posted down payments
1567     else:

```

```

1566         return False
1567
1568     return self.order_line.filtered(show_line)
1569
1570 def _get_default_payment_link_values(self):
1571     self.ensure_one()
1572     amount_max = self.amount_total - self.amount_paid
1573
1574     # Always default to the minimum value needed to confirm the order:
1575     # - order is not confirmed yet
1576     # - can be confirmed online
1577     # - we have still not paid enough for confirmation.
1578     prepayment_amount = self._get_prepayment_required_amount()
1579     if (
1580         self.state in ('draft', 'sent')
1581         and self.require_payment
1582         and self.currency_id.compare_amounts(prepayment_amount, self.amount_paid)
1583         > 0
1584     ):
1585         amount = prepayment_amount - self.amount_paid
1586     else:
1587         amount = amount_max
1588
1589     return {
1590         'currency_id': self.currency_id.id,
1591         'partner_id': self.partner_invoice_id.id,
1592         'amount': amount,
1593         'amount_max': amount_max,
1594         'amount_paid': self.amount_paid,
1595     }
1596
1597 # PORTAL #
1598
1599 def _has_to_be_signed(self):
1600     """A sale order has to be signed when:
1601     - its state is 'draft' or 'sent'
1602     - it's not expired;
1603     - it requires a signature;
1604     - it's not already signed.
1605
1606     Note: self.ensure_one()
1607
1608     :return: Whether the sale order has to be signed.
1609     :rtype: bool
1610     """
1611     self.ensure_one()
1612     return (
1613         self.state in ['draft', 'sent']
1614         and not self.is_expired
1615         and self.require_signature
1616         and not self.signature
1617     )
1618
1619 def _has_to_be_paid(self):
1620     """A sale order has to be paid when:
1621     - its state is 'draft' or 'sent';
1622     - it's not expired;
1623     - it requires a payment;
1624     - the last transaction's state isn't 'done';
1625     - the total amount is strictly positive.
1626
1627     Note: self.ensure_one()
1628
1629     :return: Whether the sale order has to be paid.
1630     :rtype: bool
1631     """
1632     self.ensure_one()
1633     transaction = self.get_portal_last_transaction()
1634     return (
1635         self.state in ['draft', 'sent']
1636         and not self.is_expired
1637         and self.require_payment

```

```

1637         and transaction.state != 'done'
1638         and self.amount_total > 0
1639     )
1640
1641     def _get_portal_return_action(self):
1642         """ Return the action used to display orders when returning from customer
        portal. """
1643         self.ensure_one()
1644         return self.env.ref('sale.action_quotations_with_onboarding')
1645
1646     def _get_name_portal_content_view(self):
1647         """ This method can be inherited by localizations who want to localize the
        online quotation view. """
1648         self.ensure_one()
1649         return 'sale.sale_order_portal_content'
1650
1651     def _get_name_tax_totals_view(self):
1652         """ This method can be inherited by localizations who want to localize the
        taxes displayed on the portal and sale order report. """
1653         return 'sale.document_tax_totals'
1654
1655     def _get_report_base_filename(self):
1656         self.ensure_one()
1657         return f'{self.type_name} {self.name}'
1658
1659     #=== CORE METHODS OVERRIDES ===#
1660
1661     @api.model
1662     def get_empty_list_help(self, help_msg):
1663         self = self.with_context(
1664             empty_list_help_document_name=_("sale order"),
1665         )
1666         return super().get_empty_list_help(help_msg)
1667
1668     def _compute_field_value(self, field):
1669         if field.name != 'invoice_status' or self.env.context.get(
1670             'mail_activity_automation_skip'):
1671             return super()._compute_field_value(field)
1672
1673         filtered_self = self.filtered(
1674             lambda so: so.ids
1675             and (so.user_id or so.partner_id.user_id)
1676             and so.origin.invoice_status != 'upselling')
1677         super()._compute_field_value(field)
1678
1679         upselling_orders = filtered_self.filtered(lambda so: so.invoice_status ==
1680             'upselling')
1681         upselling_orders._create_upsell_activity()
1682
1683     #=== BUSINESS METHODS ===#
1684
1685     def _create_upsell_activity(self):
1686         if not self:
1687             return
1688
1689         self.activity_unlink(['sale.mail_act_sale_upsell'])
1690         for order in self:
1691             order_ref = order._get_html_link()
1692             customer_ref = order.partner_id._get_html_link()
1693             order.activity_schedule(
1694                 'sale.mail_act_sale_upsell',
1695                 user_id=order.user_id.id or order.partner_id.user_id.id,
1696                 note=_(
1697                     "Upsell %(order)s for customer %(customer)s",
1698                     order=order_ref,
1699                     customer=customer_ref))
1700
1701     def _prepare_analytic_account_data(self, prefix=None):
1702         """ Prepare SO analytic account creation values.
1703
1704         :param str prefix: The prefix of the to-be-created analytic account name
1705         :return: `account.analytic.account` creation values
1706         :rtype: dict
1707         """

```

```

1703     self.ensure_one()
1704     name = self.name
1705     if prefix:
1706         name = prefix + ": " + self.name
1707     plan = self.env['account.analytic.plan'].sudo().search([], limit=1)
1708     if not plan:
1709         plan = self.env['account.analytic.plan'].sudo().create({
1710             'name': 'Default',
1711         })
1712     return {
1713         'name': name,
1714         'code': self.client_order_ref,
1715         'company_id': self.company_id.id,
1716         'plan_id': plan.id,
1717         'partner_id': self.partner_id.id,
1718     }
1719
1720 def _create_analytic_account(self, prefix=None):
1721     """ Create a new analytic account for the given orders.
1722
1723     :param str prefix: if specified, the account name will be '<prefix>:
1724     <so_reference>'.
1725     If not, the account name will be the Sales Order reference.
1726     :return: None
1727     """
1728     for order in self:
1729         analytic = self.env['account.analytic.account'].create(order._prepare_analytic_account_data(prefix))
1730         order.analytic_account_id = analytic
1731
1732 def _prepare_down_payment_section_line(self, **optional_values):
1733     """ Prepare the values to create a new down payment section.
1734
1735     :param dict optional_values: any parameter that should be added to the
1736     returned down payment section
1737     :return: `account.move.line` creation values
1738     :rtype: dict
1739     """
1740     self.ensure_one()
1741     context = {'lang': self.partner_id.lang}
1742     down_payments_section_line = {
1743         'display_type': 'line_section',
1744         'name': _("Down Payments"),
1745         'product_id': False,
1746         'product_uom_id': False,
1747         'quantity': 0,
1748         'discount': 0,
1749         'price_unit': 0,
1750         'account_id': False,
1751         **optional_values
1752     }
1753     del context
1754     return down_payments_section_line
1755
1756 def _get_prepayment_required_amount(self):
1757     """ Return the minimum amount needed to confirm automatically the quotation.
1758
1759     Note: self.ensure_one()
1760
1761     :return: The minimum amount needed to confirm automatically the quotation.
1762     :rtype: float
1763     """
1764     self.ensure_one()
1765     if self.prepayment_percent == 1.0 or not self.require_payment:
1766         return self.amount_total
1767     else:
1768         return self.currency_id.round(self.amount_total * self.prepayment_percent)
1769
1770 def _is_confirmation_amount_reached(self):
1771     """ Return whether `self.amount_paid` is higher than the prepayment required
1772     amount.

```

```

1771         Note: self.ensure_one()
1772
1773         :return: Whether `self.amount_paid` is higher than the prepayment required
1774         amount.
1775         :rtype: bool
1776         """
1777         self.ensure_one()
1778         amount_comparison = self.currency_id.compare_amounts(
1779             self._get_prepayment_required_amount(), self.amount_paid,
1780         )
1781         return amount_comparison <= 0
1782
1783     def _generate_downpayment_invoices(self):
1784         """ Generate invoices as down payments for sale order.
1785
1786         :return: The generated down payment invoices.
1787         :rtype: recordset of `account.move`
1788         """
1789         generated_invoices = self.env['account.move']
1790
1791         for order in self:
1792             downpayment_wizard = order.env['sale.advance.payment.inv'].create({
1793                 'sale_order_ids': order,
1794                 'advance_payment_method': 'fixed',
1795                 'fixed_amount': order.amount_paid,
1796             })
1797             generated_invoices |= downpayment_wizard._create_invoices(order)
1798
1799         return generated_invoices
1800
1801     def _get_product_catalog_order_data(self, products, **kwargs):
1802         pricelist = self.pricelist_id._get_products_price(
1803             quantity=1.0,
1804             products=products,
1805             currency=self.currency_id,
1806             date=self.date_order,
1807             **kwargs,
1808         )
1809         return {product_id: {'price': price} for product_id, price in pricelist.items
1810             ()}
1811
1812     def _get_product_catalog_record_lines(self, product_ids):
1813         grouped_lines = defaultdict(lambda: self.env['sale.order.line'])
1814         for line in self.order_line:
1815             if line.display_type or line.product_id.id not in product_ids:
1816                 continue
1817             grouped_lines[line.product_id] |= line
1818         return grouped_lines
1819
1820     def _get_product_documents(self):
1821         self.ensure_one()
1822
1823         documents = (
1824             self.order_line.product_id.product_document_ids
1825             | self.order_line.product_template_id.product_document_ids
1826         )
1827         return self._filter_product_documents(documents).sorted()
1828
1829     def _filter_product_documents(self, documents):
1830         return documents.filtered(
1831             lambda document:
1832                 document.attached_on == 'quotation'
1833                 or (self.state == 'sale' and document.attached_on == 'sale_order')
1834         )
1835
1836     def _update_order_line_info(self, product_id, quantity, **kwargs):
1837         """ Update sale order line information for a given product or create a
1838         new one if none exists yet.
1839         :param int product_id: The product, as a `product.product` id.
1840         :return: The unit price of the product, based on the pricelist of the
1841                 sale order and the quantity selected.
1842         :rtype: float

```

```

1841         """
1842         sol = self.order_line.filtered(lambda line: line.product_id.id == product_id)
1843         if sol:
1844             if quantity != 0:
1845                 sol.product_uom_qty = quantity
1846             elif self.state in ['draft', 'sent']:
1847                 price_unit = self.pricelist_id._get_product_price(
1848                     product=sol.product_id,
1849                     quantity=1.0,
1850                     currency=self.currency_id,
1851                     date=self.date_order,
1852                     **kwargs,
1853                 )
1854                 sol.unlink()
1855                 return price_unit
1856             else:
1857                 sol.product_uom_qty = 0
1858         elif quantity > 0:
1859             sol = self.env['sale.order.line'].create({
1860                 'order_id': self.id,
1861                 'product_id': product_id,
1862                 'product_uom_qty': quantity,
1863                 'sequence': ((self.order_line and self.order_line[-1].sequence + 1) or
1864                             10), # put it at the end of the order
1865             })
1866         return sol.price_unit
1867
1868     #=== HOOKS ===#
1869
1870     def add_option_to_order_with_taxcloud(self):
1871         self.ensure_one()
1872
1873     def validate_taxes_on_sales_order(self):
1874         # Override for correct taxcloud computation
1875         # when using coupon and delivery
1876         return True
1877
1878     #=== TOOLING ===#
1879
1880     def _is_readonly(self):
1881         """ Return Whether the sale order is read-only or not based on the state or
1882         the lock status.
1883
1884         A sale order is considered read-only if its state is 'cancel' or if the sale
1885         order is
1886         locked.
1887
1888         :return: Whether the sale order is read-only or not.
1889         :rtype: bool
1890         """
1891         self.ensure_one()
1892         return self.state == 'cancel' or self.locked
1893
1894     def _is_paid(self):
1895         """ Return whether the sale order is paid or not based on the linked
1896         transactions.
1897
1898         A sale order is considered paid if the sum of all the linked transaction is
1899         equal to or
1900         higher than `self.amount_total`.
1901
1902         :return: Whether the sale order is paid or not.
1903         :rtype: bool
1904         """
1905         self.ensure_one()
1906         return self.currency_id.compare_amounts(self.amount_paid, self.amount_total)
1907         >= 0
1908
1909 # sale_order_line.py
1910 # -*- coding: utf-8 -*-
1911 # Part of Odoo. See LICENSE file for full copyright and licensing details.

```



```

1907 from collections import defaultdict
1908 from datetime import timedelta
1909 from markupsafe import Markup
1910
1911 from odoo import api, fields, models, _
1912 from odoo.exceptions import UserError
1913 from odoo.fields import Command
1914 from odoo.osv import expression
1915 from odoo.tools import float_is_zero, float_compare, float_round, format_date, groupby
1916
1917
1918 class SaleOrderLine(models.Model):
1919     _name = 'sale.order.line'
1920     _inherit = 'analytic.mixin'
1921     _description = "Sales Order Line"
1922     _rec_names_search = ['name', 'order_id.name']
1923     _order = 'order_id, sequence, id'
1924     _check_company_auto = True
1925
1926     _sql_constraints = [
1927         ('accountable_required_fields',
1928          "CHECK(display_type IS NOT NULL OR (product_id IS NOT NULL AND
1929            product_uom IS NOT NULL))",
1930          "Missing required fields on accountable sale order line."),
1931         ('non_accountable_null_fields',
1932          "CHECK(display_type IS NULL OR (product_id IS NULL AND price_unit = 0 AND
1933            product_uom_qty = 0 AND product_uom IS NULL AND customer_lead = 0))",
1934          "Forbidden values on non-accountable sale order line"),
1935     ]
1936
1937     # Fields are ordered according by tech & business logics
1938     # and computed fields are defined after their dependencies.
1939     # This reduces execution stacks depth when precomputing fields
1940     # on record creation (and is also a good ordering logic imho)
1941
1942     order_id = fields.Many2one(
1943         comodel_name='sale.order',
1944         string="Order Reference",
1945         required=True, ondelete='cascade', index=True, copy=False)
1946     sequence = fields.Integer(string="Sequence", default=10)
1947
1948     # Order-related fields
1949     company_id = fields.Many2one(
1950         related='order_id.company_id',
1951         store=True, index=True, precompute=True)
1952     currency_id = fields.Many2one(
1953         related='order_id.currency_id',
1954         depends=['order_id.currency_id'],
1955         store=True, precompute=True)
1956     order_partner_id = fields.Many2one(
1957         related='order_id.partner_id',
1958         string="Customer",
1959         store=True, index=True, precompute=True)
1960     salesman_id = fields.Many2one(
1961         related='order_id.user_id',
1962         string="Salesperson",
1963         store=True, precompute=True)
1964     state = fields.Selection(
1965         related='order_id.state',
1966         string="Order Status",
1967         copy=False, store=True, precompute=True)
1968     tax_country_id = fields.Many2one(related='order_id.tax_country_id')
1969
1970     # Fields specifying custom line logic
1971     display_type = fields.Selection(
1972         selection=[
1973             ('line_section', "Section"),
1974             ('line_note', "Note"),
1975         ],
1976         default=False)
1977     is_downpayment = fields.Boolean(
1978         string="Is a down payment",

```

```

1977         help="Down payments are made when creating invoices from a sales order."
1978         " They are not copied when duplicating a sales order.")
1979     is_expense = fields.Boolean(
1980         string="Is expense",
1981         help="Is true if the sales order line comes from an expense or a vendor bills"
1982     )
1983
1984     # Generic configuration fields
1985     product_id = fields.Many2one(
1986         comodel_name='product.product',
1987         string="Product",
1988         change_default=True, ondelete='restrict', check_company=True, index=
1989         'btree_not_null',
1990         domain=[('sale_ok', '=', True)])
1991     product_template_id = fields.Many2one(
1992         string="Product Template",
1993         comodel_name='product.template',
1994         compute='_compute_product_template_id',
1995         readonly=False,
1996         search='_search_product_template_id',
1997         # previously related='product_id.product_tmpl_id'
1998         # not anymore since the field must be considered editable for product
1999         # configurator logic
2000         # without modifying the related product_id when updated.
2001         domain=[('sale_ok', '=', True)])
2002     product_uom_category_id = fields.Many2one(related='product_id.uom_id.category_id',
2003         depends=['product_id'])
2004
2005     product_custom_attribute_value_ids = fields.One2many(
2006         comodel_name='product.attribute.custom.value', inverse_name=
2007         'sale_order_line_id',
2008         string="Custom Values",
2009         compute='_compute_custom_attribute_values',
2010         store=True, readonly=False, precompute=True, copy=True)
2011     # M2M holding the values of product.attribute with create_variant field set to
2012     # 'no_variant'
2013     # It allows keeping track of the extra_price associated to those attribute values
2014     # and add them to the SO line description
2015     product_no_variant_attribute_value_ids = fields.Many2many(
2016         comodel_name='product.template.attribute.value',
2017         string="Extra Values",
2018         compute='_compute_no_variant_attribute_values',
2019         store=True, readonly=False, precompute=True, ondelete='restrict')
2020
2021     name = fields.Text(
2022         string="Description",
2023         compute='_compute_name',
2024         store=True, readonly=False, required=True, precompute=True)
2025
2026     product_uom_qty = fields.Float(
2027         string="Quantity",
2028         compute='_compute_product_uom_qty',
2029         digits='Product Unit of Measure', default=1.0,
2030         store=True, readonly=False, required=True, precompute=True)
2031     product_uom = fields.Many2one(
2032         comodel_name='uom.uom',
2033         string="Unit of Measure",
2034         compute='_compute_product_uom',
2035         store=True, readonly=False, precompute=True, ondelete='restrict',
2036         domain=[('category_id', '=', product_uom_category_id)])
2037
2038     # Pricing fields
2039     tax_id = fields.Many2many(
2040         comodel_name='account.tax',
2041         string="Taxes",
2042         compute='_compute_tax_id',
2043         store=True, readonly=False, precompute=True,
2044         context={'active_test': False},
2045         check_company=True)
2046
2047     # Tech field caching pricelist rule used for price & discount computation
2048     pricelist_item_id = fields.Many2one(

```

```

2042         comodel_name='product.pricelist.item',
2043         compute='_compute_pricelist_item_id')
2044
2045     price_unit = fields.Float(
2046         string="Unit Price",
2047         compute='_compute_price_unit',
2048         digits='Product Price',
2049         store=True, readonly=False, required=True, precompute=True)
2050
2051     discount = fields.Float(
2052         string="Discount (%)",
2053         compute='_compute_discount',
2054         digits='Discount',
2055         store=True, readonly=False, precompute=True)
2056
2057     price_subtotal = fields.Monetary(
2058         string="Subtotal",
2059         compute='_compute_amount',
2060         store=True, precompute=True)
2061     price_tax = fields.Float(
2062         string="Total Tax",
2063         compute='_compute_amount',
2064         store=True, precompute=True)
2065     price_total = fields.Monetary(
2066         string="Total",
2067         compute='_compute_amount',
2068         store=True, precompute=True)
2069     price_reduce_taxexcl = fields.Monetary(
2070         string="Price Reduce Tax excl",
2071         compute='_compute_price_reduce_taxexcl',
2072         store=True, precompute=True)
2073     price_reduce_taxinc = fields.Monetary(
2074         string="Price Reduce Tax incl",
2075         compute='_compute_price_reduce_taxinc',
2076         store=True, precompute=True)
2077
2078     # Logistics/Delivery fields
2079     product_packaging_id = fields.Many2one(
2080         comodel_name='product.packaging',
2081         string="Packaging",
2082         compute='_compute_product_packaging_id',
2083         store=True, readonly=False, precompute=True,
2084         domain="[('sales', '=', True), ('product_id', '=', product_id)]",
2085         check_company=True)
2086     product_packaging_qty = fields.Float(
2087         string="Packaging Quantity",
2088         compute='_compute_product_packaging_qty',
2089         store=True, readonly=False, precompute=True)
2090
2091     customer_lead = fields.Float(
2092         string="Lead Time",
2093         compute='_compute_customer_lead',
2094         store=True, readonly=False, required=True, precompute=True,
2095         help="Number of days between the order confirmation and the shipping of the
                products to the customer")
2096
2097     qty_delivered_method = fields.Selection(
2098         selection=[
2099             ('manual', "Manual"),
2100             ('analytic', "Analytic From Expenses"),
2101         ],
2102         string="Method to update delivered qty",
2103         compute='_compute_qty_delivered_method',
2104         store=True, precompute=True,
2105         help="According to product configuration, the delivered quantity can be
                automatically computed by mechanism:\n"
                " - Manual: the quantity is set manually on the line\n"
                " - Analytic From expenses: the quantity is the quantity sum from
                posted expenses\n"
                " - Timesheet: the quantity is the sum of hours recorded on tasks
                linked to this sale line\n"
                " - Stock Moves: the quantity comes from confirmed pickings\n")
2106
2107
2108
2109

```

```

2110 qty_delivered = fields.Float(
2111     string="Delivery Quantity",
2112     compute='_compute_qty_delivered',
2113     digits='Product Unit of Measure',
2114     store=True, readonly=False, copy=False)
2115
2116 # Analytic & Invoicing fields
2117 qty_invoiced = fields.Float(
2118     string="Invoiced Quantity",
2119     compute='_compute_qty_invoiced',
2120     digits='Product Unit of Measure',
2121     store=True)
2122 qty_to_invoice = fields.Float(
2123     string="Quantity To Invoice",
2124     compute='_compute_qty_to_invoice',
2125     digits='Product Unit of Measure',
2126     store=True)
2127
2128 analytic_line_ids = fields.One2many(
2129     comodel_name='account.analytic.line', inverse_name='so_line',
2130     string="Analytic lines")
2131
2132 invoice_lines = fields.Many2many(
2133     comodel_name='account.move.line',
2134     relation='sale_order_line_invoice_rel', column1='order_line_id', column2=
2135         'invoice_line_id',
2136     string="Invoice Lines",
2137     copy=False)
2138 invoice_status = fields.Selection(
2139     selection=[
2140         ('upselling', "Upselling Opportunity"),
2141         ('invoiced', "Fully Invoiced"),
2142         ('to invoice', "To Invoice"),
2143         ('no', "Nothing to Invoice"),
2144     ],
2145     string="Invoice Status",
2146     compute='_compute_invoice_status',
2147     store=True)
2148
2149 untaxed_amount_invoiced = fields.Monetary(
2150     string="Untaxed Invoiced Amount",
2151     compute='_compute_untaxed_amount_invoiced',
2152     store=True)
2153 untaxed_amount_to_invoice = fields.Monetary(
2154     string="Untaxed Amount To Invoice",
2155     compute='_compute_untaxed_amount_to_invoice',
2156     store=True)
2157
2158 # Technical computed fields for UX purposes (hide/make fields readonly, ...)
2159 product_type = fields.Selection(related='product_id.detailed_type', depends=[
2160     'product_id'])
2161 product_updatable = fields.Boolean(
2162     string="Can Edit Product",
2163     compute='_compute_product_updatable')
2164 product_uom_readonly = fields.Boolean(
2165     compute='_compute_product_uom_readonly')
2166 tax_calculation_rounding_method = fields.Selection(
2167     related='company_id.tax_calculation_rounding_method',
2168     string='Tax calculation rounding method', readonly=True)
2169
2170 #=== COMPUTE METHODS ===#
2171
2172 @api.depends('order_partner_id', 'order_id', 'product_id')
2173 def _compute_display_name(self):
2174     name_per_id = self._additional_name_per_id()
2175     for so_line in self.sudo():
2176         name = '{} - {}'.format(so_line.order_id.name, so_line.name and so_line.
2177             name.split('\n')[0] or so_line.product_id.name)
2178         additional_name = name_per_id.get(so_line.id)
2179         if additional_name:
2180             name = f'{name} {additional_name}'
2181     so_line.display_name = name

```

```

2179
2180 @api.depends('product_id')
2181 def _compute_product_template_id(self):
2182     for line in self:
2183         line.product_template_id = line.product_id.product_tmpl_id
2184
2185 def _search_product_template_id(self, operator, value):
2186     return [('product_id.product_tmpl_id', operator, value)]
2187
2188 @api.depends('product_id')
2189 def _compute_custom_attribute_values(self):
2190     for line in self:
2191         if not line.product_id:
2192             line.product_custom_attribute_value_ids = False
2193             continue
2194         if not line.product_custom_attribute_value_ids:
2195             continue
2196         valid_values = line.product_id.product_tmpl_id.
2197         valid_product_template_attribute_line_ids.product_template_value_ids
2198         # remove the is_custom values that don't belong to this template
2199         for pacv in line.product_custom_attribute_value_ids:
2200             if pacv.custom_product_template_attribute_value_id not in valid_values
2201             :
2202                 line.product_custom_attribute_value_ids -= pacv
2203
2204 @api.depends('product_id')
2205 def _compute_no_variant_attribute_values(self):
2206     for line in self:
2207         if not line.product_id:
2208             line.product_no_variant_attribute_value_ids = False
2209             continue
2210         if not line.product_no_variant_attribute_value_ids:
2211             continue
2212         valid_values = line.product_id.product_tmpl_id.
2213         valid_product_template_attribute_line_ids.product_template_value_ids
2214         # remove the no_variant attributes that don't belong to this template
2215         for ptav in line.product_no_variant_attribute_value_ids:
2216             if ptav._origin not in valid_values:
2217                 line.product_no_variant_attribute_value_ids -= ptav
2218
2219 @api.depends('product_id')
2220 def _compute_name(self):
2221     for line in self:
2222         if not line.product_id:
2223             continue
2224         if not line.order_partner_id.is_public:
2225             line = line.with_context(lang=line.order_partner_id.lang)
2226             name = line._get_sale_order_line_multiline_description_sale()
2227             if line.is_downpayment and not line.display_type:
2228                 context = {'lang': line.order_partner_id.lang}
2229                 dp_state = line._get_downpayment_state()
2230                 if dp_state == 'draft':
2231                     name = _("%(line_description)s (Draft)", line_description=name)
2232                 elif dp_state == 'cancel':
2233                     name = _("%(line_description)s (Canceled)", line_description=name)
2234                 else:
2235                     invoice = line._get_invoice_lines().move_id
2236                     if len(invoice) == 1 and invoice.payment_reference and invoice.
2237                     invoice_date:
2238                         name = _(
2239                             "%(line_description)s (ref: %(reference)s on %(date)s)",
2240                             line_description=name,
2241                             reference=invoice.payment_reference,
2242                             date=format_date(line.env, invoice.invoice_date),
2243                         )
2244                     del context
2245             line.name = name
2246
2247 def _get_sale_order_line_multiline_description_sale(self):
2248     """ Compute a default multiline description for this sales order line.
2249
2250     In most cases the product description is enough but sometimes we need to

```

```

2247         append information that only
2248         exists on the sale order line itself.
2249         e.g:
2250         - custom attributes and attributes that don't create variants, both
          introduced by the "product configurator"
2251         - in event_sale we need to know specifically the sales order line as well as
          the product to generate the name:
2252         the product is not sufficient because we also need to know the event_id and
2253         the event_ticket_id (both which belong to the sale order line).
2254         """
2255         self.ensure_one()
2256         return self.product_id.get_product_multiline_description_sale() + self.
          _get_sale_order_line_multiline_description_variants()
2257
2258     def _get_sale_order_line_multiline_description_variants(self):
2259         """When using no_variant attributes or is_custom values, the product
2260         itself is not sufficient to create the description: we need to add
2261         information about those special attributes and values.
2262
2263         :return: the description related to special variant attributes/values
2264         :rtype: string
2265         """
2266         if not self.product_custom_attribute_value_ids and not self.
          product_no_variant_attribute_value_ids:
2267             return ""
2268
2269         name = ""
2270
2271         custom_ptavs = self.product_custom_attribute_value_ids.
          custom_product_template_attribute_value_id
2272         no_variant_ptavs = self.product_no_variant_attribute_value_ids._origin
2273         multi_ptavs = no_variant_ptavs.filtered(lambda ptav: ptav.display_type ==
          'multi').sorted()
2274
2275         # display the no_variant attributes, except those that are also
2276         # displayed by a custom (avoid duplicate description)
2277         for ptav in (no_variant_ptavs - multi_ptavs - custom_ptavs):
2278             name += "\n" + ptav.display_name
2279
2280         # display the selected values per attribute on a single for a multi checkbox
2281         for pta, ptavs in groupby(multi_ptavs, lambda ptav: ptav.attribute_id):
2282             name += "\n" + _ (
2283                 "%(attribute)s: %(values)s",
2284                 attribute=pta.name,
2285                 values=", ".join(ptav.name for ptav in ptavs)
2286             )
2287
2288         # Sort the values according to _order settings, because it doesn't work for
2289         virtual records in onchange
2290         sorted_custom_ptav = self.product_custom_attribute_value_ids.
          custom_product_template_attribute_value_id.sorted()
2291         for patv in sorted_custom_ptav:
2292             pacv = self.product_custom_attribute_value_ids.filtered(lambda pcav: pcav.
          custom_product_template_attribute_value_id == patv)
2293             name += "\n" + pacv.display_name
2294
2295         return name
2296
2297     @api.depends('display_type', 'product_id', 'product_packaging_qty')
2298     def _compute_product_uom_qty(self):
2299         for line in self:
2300             if line.display_type:
2301                 line.product_uom_qty = 0.0
2302                 continue
2303
2304             if not line.product_packaging_id:
2305                 continue
2306             packaging_uom = line.product_packaging_id.product_uom_id
2307             qty_per_packaging = line.product_packaging_id.qty
2308             product_uom_qty = packaging_uom._compute_quantity(
2309                 line.product_packaging_qty * qty_per_packaging, line.product_uom)
2310             if float_compare(product_uom_qty, line.product_uom_qty, precision_rounding

```

```

        =line.product_uom.rounding) != 0:
2308         line.product_uom_qty = product_uom_qty
2309
2310     @api.depends('product_id')
2311     def _compute_product_uom(self):
2312         for line in self:
2313             if not line.product_uom or (line.product_id.uom_id.id != line.product_uom.
                id):
2314                 line.product_uom = line.product_id.uom_id
2315
2316     @api.depends('product_id', 'company_id')
2317     def _compute_tax_id(self):
2318         taxes_by_product_company = defaultdict(lambda: self.env['account.tax'])
2319         lines_by_company = defaultdict(lambda: self.env['sale.order.line'])
2320         cached_taxes = {}
2321         for line in self:
2322             lines_by_company[line.company_id] += line
2323         for product in self.product_id:
2324             for tax in product.taxes_id:
2325                 taxes_by_product_company[(product, tax.company_id)] += tax
2326         for company, lines in lines_by_company.items():
2327             for line in lines.with_company(company):
2328                 taxes, comp = None, company
2329                 while not taxes and comp:
2330                     taxes = taxes_by_product_company[(line.product_id, comp)]
2331                     comp = comp.parent_id
2332                 if not line.product_id or not taxes:
2333                     # Nothing to map
2334                     line.tax_id = False
2335                     continue
2336                 fiscal_position = line.order_id.fiscal_position_id
2337                 cache_key = (fiscal_position.id, company.id, tuple(taxes.ids))
2338                 if cache_key in cached_taxes:
2339                     result = cached_taxes[cache_key]
2340                 else:
2341                     result = fiscal_position.map_tax(taxes)
2342                     cached_taxes[cache_key] = result
2343                 # If company_id is set, always filter taxes by the company
2344                 line.tax_id = result
2345
2346     @api.depends('product_id', 'product_uom', 'product_uom_qty')
2347     def _compute_pricelist_item_id(self):
2348         for line in self:
2349             if not line.product_id or line.display_type or not line.order_id.
                pricelist_id:
2350                 line.pricelist_item_id = False
2351             else:
2352                 line.pricelist_item_id = line.order_id.pricelist_id._get_product_rule(
2353                     line.product_id,
2354                     quantity=line.product_uom_qty or 1.0,
2355                     uom=line.product_uom,
2356                     date=line.order_id.date_order,
2357                 )
2358
2359     @api.depends('product_id', 'product_uom', 'product_uom_qty')
2360     def _compute_price_unit(self):
2361         for line in self:
2362             # check if there is already invoiced amount. if so, the price shouldn't
2363             # change as it might have been
2364             # manually edited
2365             if line.qty_invoiced > 0:
2366                 continue
2367             if not line.product_uom or not line.product_id:
2368                 line.price_unit = 0.0
2369             else:
2370                 price = line.with_company(line.company_id)._get_display_price()
2371                 line.price_unit = line.product_id._get_tax_included_unit_price(
2372                     line.company_id or line.env.company,
2373                     line.order_id.currency_id,
2374                     line.order_id.date_order,
2375                     'sale',
2376                     fiscal_position=line.order_id.fiscal_position_id,

```



```

2376         product_price_unit=price,
2377         product_currency=line.currency_id
2378     )
2379
2380 def _get_display_price(self):
2381     """Compute the displayed unit price for a given line.
2382
2383     Overridden in custom flows:
2384     * where the price is not specified by the pricelist
2385     * where the discount is not specified by the pricelist
2386
2387     Note: self.ensure_one()
2388     """
2389     self.ensure_one()
2390
2391     pricelist_price = self._get_pricelist_price()
2392
2393     if self.order_id.pricelist_id.discount_policy == 'with_discount':
2394         return pricelist_price
2395
2396     if not self.pricelist_item_id:
2397         # No pricelist rule found => no discount from pricelist
2398         return pricelist_price
2399
2400     base_price = self._get_pricelist_price_before_discount()
2401
2402     # negative discounts (= surcharge) are included in the display price
2403     return max(base_price, pricelist_price)
2404
2405 def _get_pricelist_price(self):
2406     """Compute the price given by the pricelist for the given line information.
2407
2408     :return: the product sales price in the order currency (without taxes)
2409     :rtype: float
2410     """
2411     self.ensure_one()
2412     self.product_id.ensure_one()
2413
2414     price = self.pricelist_item_id._compute_price(
2415         product=self.product_id.with_context(**self._get_product_price_context()),
2416         quantity=self.product_uom_qty or 1.0,
2417         uom=self.product_uom,
2418         date=self.order_id.date_order,
2419         currency=self.currency_id,
2420     )
2421
2422     return price
2423
2424 def _get_product_price_context(self):
2425     """Gives the context for product price computation.
2426
2427     :return: additional context to consider extra prices from attributes in the
2428     base product price.
2429     :rtype: dict
2430     """
2431     self.ensure_one()
2432     return self.product_id._get_product_price_context(
2433         self.product_no_variant_attribute_value_ids,
2434     )
2435
2436 def _get_pricelist_price_context(self):
2437     """DO NOT USE in new code, this contextual logic should be dropped or heavily
2438     refactored soon"""
2439     self.ensure_one()
2440     return {
2441         'pricelist': self.order_id.pricelist_id.id,
2442         'uom': self.product_uom.id,
2443         'quantity': self.product_uom_qty,
2444         'date': self.order_id.date_order,
2445     }
2446
2447 def _get_pricelist_price_before_discount(self):

```

```

2446         """Compute the price used as base for the pricelist price computation.
2447
2448         :return: the product sales price in the order currency (without taxes)
2449         :rtype: float
2450         """
2451         self.ensure_one()
2452         self.product_id.ensure_one()
2453
2454         return self.pricelist_item_id.compute_price_before_discount(
2455             product=self.product_id.with_context(**self._get_product_price_context()),
2456             quantity=self.product_uom_qty or 1.0,
2457             uom=self.product_uom,
2458             date=self.order_id.date_order,
2459             currency=self.currency_id,
2460         )
2461
2462     @api.depends('product_id', 'product_uom', 'product_uom_qty')
2463     def _compute_discount(self):
2464         for line in self:
2465             if not line.product_id or line.display_type:
2466                 line.discount = 0.0
2467
2468             if not (
2469                 line.order_id.pricelist_id
2470                 and line.order_id.pricelist_id.discount_policy == 'without_discount'
2471             ):
2472                 continue
2473
2474             line.discount = 0.0
2475
2476             if not line.pricelist_item_id:
2477                 # No pricelist rule was found for the product
2478                 # therefore, the pricelist didn't apply any discount/change
2479                 # to the existing sales price.
2480                 continue
2481
2482             line = line.with_company(line.company_id)
2483             pricelist_price = line._get_pricelist_price()
2484             base_price = line._get_pricelist_price_before_discount()
2485
2486             if base_price != 0: # Avoid division by zero
2487                 discount = (base_price - pricelist_price) / base_price * 100
2488                 if (discount > 0 and base_price > 0) or (discount < 0 and base_price <
2489                     0):
2490                     # only show negative discounts if price is negative
2491                     # otherwise it's a surcharge which shouldn't be shown to the
2492                     # customer
2493                     line.discount = discount
2494
2495     def _convert_to_tax_base_line_dict(self, **kwargs):
2496         """ Convert the current record to a dictionary in order to use the generic
2497         taxes computation method
2498         defined on account.tax.
2499
2500         :return: A python dictionary.
2501         """
2502         self.ensure_one()
2503         return self.env['account.tax']._convert_to_tax_base_line_dict(
2504             self,
2505             partner=self.order_id.partner_id,
2506             currency=self.order_id.currency_id,
2507             product=self.product_id,
2508             taxes=self.tax_id,
2509             price_unit=self.price_unit,
2510             quantity=self.product_uom_qty,
2511             discount=self.discount,
2512             price_subtotal=self.price_subtotal,
2513             **kwargs,
2514         )
2515
2516     @api.depends('product_uom_qty', 'discount', 'price_unit', 'tax_id')
2517     def _compute_amount(self):

```

```

2515         """
2516         Compute the amounts of the SO line.
2517         """
2518         for line in self:
2519             tax_results = self.env['account.tax']._compute_taxes([
2520                 line._convert_to_tax_base_line_dict()
2521             ])
2522             totals = list(tax_results['totals'].values())[0]
2523             amount_untaxed = totals['amount_untaxed']
2524             amount_tax = totals['amount_tax']
2525
2526             line.update({
2527                 'price_subtotal': amount_untaxed,
2528                 'price_tax': amount_tax,
2529                 'price_total': amount_untaxed + amount_tax,
2530             })
2531
2532         @api.depends('price_subtotal', 'product_uom_qty')
2533         def _compute_price_reduce_taxexcl(self):
2534             for line in self:
2535                 line.price_reduce_taxexcl = line.price_subtotal / line.product_uom_qty if
2536                 line.product_uom_qty else 0.0
2537
2538         @api.depends('price_total', 'product_uom_qty')
2539         def _compute_price_reduce_taxinc(self):
2540             for line in self:
2541                 line.price_reduce_taxinc = line.price_total / line.product_uom_qty if line
2542                 .product_uom_qty else 0.0
2543
2544         @api.depends('product_id', 'product_uom_qty', 'product_uom')
2545         def _compute_product_packaging_id(self):
2546             for line in self:
2547                 # remove packaging if not match the product
2548                 if line.product_packaging_id.product_id != line.product_id:
2549                     line.product_packaging_id = False
2550                 # suggest biggest suitable packaging matching the SO's company
2551                 if line.product_id and line.product_uom_qty and line.product_uom:
2552                     suggested_packaging = line.product_id.packaging_ids\
2553                         .filtered(lambda p: p.sales and (p.product_id.company_id <= p.
2554                             company_id <= line.company_id))\
2555                         .find_suitable_product_packaging(line.product_uom_qty, line.
2556                             product_uom)
2557                     line.product_packaging_id = suggested_packaging or line.
2558                     product_packaging_id
2559
2560         @api.depends('product_packaging_id', 'product_uom', 'product_uom_qty')
2561         def _compute_product_packaging_qty(self):
2562             self.product_packaging_qty = 0
2563             for line in self:
2564                 if not line.product_packaging_id:
2565                     continue
2566                 line.product_packaging_qty = line.product_packaging_id._compute_qty(line.
2567                     product_uom_qty, line.product_uom)
2568
2569         # This computed default is necessary to have a clean computation inheritance
2570         # (cf sale_stock) instead of simply removing the default and specifying
2571         # the compute attribute & method in sale_stock.
2572         def _compute_customer_lead(self):
2573             self.customer_lead = 0.0
2574
2575         @api.depends('is_expense')
2576         def _compute_qty_delivered_method(self):
2577             """ Sale module compute delivered qty for product [('type', 'in', ['consu']),
2578                 ('service_type', '=', 'manual')]
2579                 - consu + expense_policy : analytic (sum of analytic unit_amount)
2580                 - consu + no expense_policy : manual (set manually on SOL)
2581                 - service (+ service_type='manual', the only available option) :
2582                     manual
2583
2584             This is true when only sale is installed: sale_stock redefine the
2585             behavior for 'consu' type,
2586             and sale_timesheet implements the behavior of 'service' +

```

```

        service_type=timesheet.
2578
2579 for line in self:
2580     if line.is_expense:
2581         line.qty_delivered_method = 'analytic'
2582     else: # service and consu
2583         line.qty_delivered_method = 'manual'
2584
2585 @api.depends(
2586     'qty_delivered_method',
2587     'analytic_line_ids.so_line',
2588     'analytic_line_ids.unit_amount',
2589     'analytic_line_ids.product_uom_id')
2590 def _compute_qty_delivered(self):
2591     """ This method compute the delivered quantity of the SO lines: it covers the
2592     case provide by sale module, aka
2593     expense/vendor bills (sum of unit_amount of AAL), and manual case.
2594     This method should be overridden to provide other way to automatically
2595     compute delivered qty. Overrides should
2596     take their concerned so lines, compute and set the `qty_delivered` field,
2597     and call super with the remaining
2598     records.
2599
2600     """
2601     # compute for analytic lines
2602     lines_by_analytic = self.filtered(lambda sol: sol.qty_delivered_method ==
2603     'analytic')
2604     mapping = lines_by_analytic._get_delivered_quantity_by_analytic([('amount',
2605     '<=', 0.0)])
2606     for so_line in lines_by_analytic:
2607         so_line.qty_delivered = mapping.get(so_line.id or so_line._origin.id, 0.0)
2608
2609 def _get_downpayment_state(self):
2610     self.ensure_one()
2611
2612     if self.display_type:
2613         return ''
2614
2615     invoice_lines = self._get_invoice_lines()
2616     if all(line.parent_state == 'draft' for line in invoice_lines):
2617         return 'draft'
2618     if all(line.parent_state == 'cancel' for line in invoice_lines):
2619         return 'cancel'
2620
2621     return ''
2622
2623 def _get_delivered_quantity_by_analytic(self, additional_domain):
2624     """ Compute and write the delivered quantity of current SO lines, based on
2625     their related
2626     analytic lines.
2627     :param additional_domain: domain to restrict AAL to include in
2628     computation (required since timesheet is an AAL with a project ...)
2629
2630     """
2631     result = defaultdict(float)
2632
2633     # avoid recomputation if no SO lines concerned
2634     if not self:
2635         return result
2636
2637     # group analytic lines by product uom and so line
2638     domain = expression.AND([(['so_line', 'in', self.ids]), additional_domain])
2639     data = self.env['account.analytic.line']._read_group(
2640         domain,
2641         ['product_uom_id', 'so_line'],
2642         ['unit_amount:sum', 'move_line_id:count_distinct', '__count'],
2643     )
2644
2645     # convert uom and sum all unit_amount of analytic lines to get the delivered
2646     qty of SO lines
2647     for uom, so_line, unit_amount_sum, move_line_id_count_distinct, count in data:
2648         if not uom:
2649             continue
2650         # avoid counting unit_amount twice when dealing with multiple analytic

```

```

        lines on the same move line
2641     if move_line_id_count_distinct == 1 and count > 1:
2642         qty = unit_amount_sum / count
2643     else:
2644         qty = unit_amount_sum
2645     if so_line.product_uom.category_id == uom.category_id:
2646         qty = uom._compute_quantity(qty, so_line.product_uom, rounding_method=
            'HALF-UP')
2647     result[so_line.id] += qty
2648
2649     return result
2650
2651 @api.depends('invoice_lines.move_id.state', 'invoice_lines.quantity')
2652 def _compute_qty_invoiced(self):
2653     """
2654     Compute the quantity invoiced. If case of a refund, the quantity invoiced is
2655     decreased. Note
2656     that this is the case only if the refund is generated from the SO and that is
2657     intentional: if
2658     a refund made would automatically decrease the invoiced quantity, then there
2659     is a risk of re-invoicing
2660     it automatically, which may not be wanted at all. That's why the refund has
2661     to be created from the SO
2662     """
2663     for line in self:
2664         qty_invoiced = 0.0
2665         for invoice_line in line._get_invoice_lines():
2666             if invoice_line.move_id.state != 'cancel' or invoice_line.move_id.
                payment_state == 'invoicing_legacy':
2667                 if invoice_line.move_id.move_type == 'out_invoice':
2668                     qty_invoiced += invoice_line.product_uom_id._compute_quantity(
                        invoice_line.quantity, line.product_uom)
2669                 elif invoice_line.move_id.move_type == 'out_refund':
2670                     qty_invoiced -= invoice_line.product_uom_id._compute_quantity(
                        invoice_line.quantity, line.product_uom)
2671         line.qty_invoiced = qty_invoiced
2672
2673 def _get_invoice_lines(self):
2674     self.ensure_one()
2675     if self._context.get('accrual_entry_date'):
2676         return self.invoice_lines.filtered(
2677             lambda l: l.move_id.invoice_date and l.move_id.invoice_date <= self.
                _context['accrual_entry_date']
2678         )
2679     else:
2680         return self.invoice_lines
2681
2682 # no trigger product_id.invoice_policy to avoid retroactively changing SO
2683 @api.depends('qty_invoiced', 'qty_delivered', 'product_uom_qty', 'state')
2684 def _compute_qty_to_invoice(self):
2685     """
2686     Compute the quantity to invoice. If the invoice policy is order, the quantity
2687     to invoice is
2688     calculated from the ordered quantity. Otherwise, the quantity delivered is
2689     used.
2690     """
2691     for line in self:
2692         if line.state == 'sale' and not line.display_type:
2693             if line.product_id.invoice_policy == 'order':
2694                 line.qty_to_invoice = line.product_uom_qty - line.qty_invoiced
2695             else:
2696                 line.qty_to_invoice = line.qty_delivered - line.qty_invoiced
2697         else:
2698             line.qty_to_invoice = 0
2699
2700 @api.depends('state', 'product_uom_qty', 'qty_delivered', 'qty_to_invoice',
    'qty_invoiced')
2701 def _compute_invoice_status(self):
2702     """
2703     Compute the invoice status of a SO line. Possible statuses:
2704     - no: if the SO is not in status 'sale', we consider that there is nothing to
2705     invoice. This is also the default value if the conditions of no other

```

```

        status is met.
2700 - to invoice: we refer to the quantity to invoice of the line. Refer to method
2701 `_compute_qty_to_invoice()` for more information on how this quantity is
        calculated.
2702 - upselling: this is possible only for a product invoiced on ordered
        quantities for which
2703 we delivered more than expected. The could arise if, for example, a project
        took more
2704 time than expected but we decided not to invoice the extra cost to the
        client. This
2705 occurs only in state 'sale', the upselling opportunity is removed from the
        list.
2706 - invoiced: the quantity invoiced is larger or equal to the quantity ordered.
2707 """
2708 precision = self.env['decimal.precision'].precision_get('Product Unit of
        Measure')
2709 for line in self:
2710     if line.state != 'sale':
2711         line.invoice_status = 'no'
2712     elif line.is_downpayment and line.untaxed_amount_to_invoice == 0:
2713         line.invoice_status = 'invoiced'
2714     elif not float_is_zero(line.qty_to_invoice, precision_digits=precision):
2715         line.invoice_status = 'to invoice'
2716     elif line.state == 'sale' and line.product_id.invoice_policy == 'order'
        and\
2717         line.product_uom_qty >= 0.0 and\
2718         float_compare(line.qty_delivered, line.product_uom_qty,
            precision_digits=precision) == 1:
2719         line.invoice_status = 'upselling'
2720     elif float_compare(line.qty_invoiced, line.product_uom_qty,
        precision_digits=precision) >= 0:
2721         line.invoice_status = 'invoiced'
2722     else:
2723         line.invoice_status = 'no'
2724
2725 @api.depends('invoice_lines', 'invoice_lines.price_total',
        'invoice_lines.move_id.state', 'invoice_lines.move_id.move_type')
2726 def _compute_untaxed_amount_invoiced(self):
2727     """ Compute the untaxed amount already invoiced from the sale order line,
        taking the refund attached
2728     the so line into account. This amount is computed as
2729     SUM(inv_line.price_subtotal) - SUM(ref_line.price_subtotal)
2730     where
2731     `inv_line` is a customer invoice line linked to the SO line
2732     `ref_line` is a customer credit note (refund) line linked to the SO
        line
2733     """
2734     for line in self:
2735         amount_invoiced = 0.0
2736         for invoice_line in line.get_invoice_lines():
2737             if invoice_line.move_id.state == 'posted':
2738                 invoice_date = invoice_line.move_id.invoice_date or fields.Date.
                    today()
2739                 if invoice_line.move_id.move_type == 'out_invoice':
2740                     amount_invoiced += invoice_line.currency_id.convert(
                        invoice_line.price_subtotal, line.currency_id, line.company_id
                        , invoice_date)
2741                 elif invoice_line.move_id.move_type == 'out_refund':
2742                     amount_invoiced -= invoice_line.currency_id.convert(
                        invoice_line.price_subtotal, line.currency_id, line.company_id
                        , invoice_date)
2743                 line.untaxed_amount_invoiced = amount_invoiced
2744
2745 @api.depends('state', 'product_id', 'untaxed_amount_invoiced', 'qty_delivered',
        'product_uom_qty', 'price_unit')
2746 def _compute_untaxed_amount_to_invoice(self):
2747     """ Total of remaining amount to invoice on the sale order line (taxes excl.)
        as
2748         total_sol - amount already invoiced
2749     where Total_sol depends on the invoice policy of the product.
2750
2751     Note: Draft invoice are ignored on purpose, the 'to invoice' amount should

```

```

2752         come only from the SO lines.
2753     """
2754     for line in self:
2755         amount_to_invoice = 0.0
2756         if line.state == 'sale':
2757             # Note: do not use price_subtotal field as it returns zero when the
2758             # ordered quantity is
2759             # zero. It causes problem for expense line (e.i.: ordered qty = 0,
2760             # deli qty = 4,
2761             # price_unit = 20 ; subtotal is zero), but when you can invoice the
2762             # line, you see an
2763             # amount and not zero. Since we compute untaxed amount, we can use
2764             # directly the price
2765             # reduce (to include discount) without using `compute_all()` method
2766             # on taxes.
2767             price_subtotal = 0.0
2768             uom_qty_to_consider = line.qty_delivered if line.product_id.
2769             invoice_policy == 'delivery' else line.product_uom_qty
2770             price_reduce = line.price_unit * (1 - (line.discount or 0.0) / 100.0)
2771             price_subtotal = price_reduce * uom_qty_to_consider
2772             if len(line.tax_id.filtered(lambda tax: tax.price_include)) > 0:
2773                 # As included taxes are not excluded from the computed subtotal,
2774                 # `compute_all()` method
2775                 # has to be called to retrieve the subtotal without them.
2776                 # `price_reduce_taxexcl` cannot be used as it is computed from
2777                 # `price_subtotal` field. (see upper Note)
2778                 price_subtotal = line.tax_id.compute_all(
2779                     price_reduce,
2780                     currency=line.currency_id,
2781                     quantity=uom_qty_to_consider,
2782                     product=line.product_id,
2783                     partner=line.order_id.partner_shipping_id)['total_excluded']
2784             inv_lines = line._get_invoice_lines()
2785             if any(inv_lines.mapped(lambda l: l.discount != line.discount)):
2786                 # In case of re-invoicing with different discount we try to
2787                 # calculate manually the
2788                 # remaining amount to invoice
2789                 amount = 0
2790                 for l in inv_lines:
2791                     if len(l.tax_ids.filtered(lambda tax: tax.price_include)) > 0:
2792                         amount += l.tax_ids.compute_all(l.currency_id._convert(l.
2793                         price_unit, line.currency_id, line.company_id, l.date or
2794                         fields.Date.today(), round=False) * l.quantity)['
2795                         total_excluded']
2796                     else:
2797                         amount += l.currency_id._convert(l.price_unit, line.
2798                         currency_id, line.company_id, l.date or fields.Date.today
2799                         (), round=False) * l.quantity
2800
2801             amount_to_invoice = max(price_subtotal - amount, 0)
2802         else:
2803             amount_to_invoice = price_subtotal - line.untaxed_amount_invoiced
2804
2805         line.untaxed_amount_to_invoice = amount_to_invoice
2806
2807     @api.depends('order_id.partner_id', 'product_id')
2808     def _compute_analytic_distribution(self):
2809         for line in self:
2810             if not line.display_type:
2811                 distribution = line.env['account.analytic.distribution.model'].
2812                 _get_distribution({
2813                     "product_id": line.product_id.id,
2814                     "product_categ_id": line.product_id.categ_id.id,
2815                     "partner_id": line.order_id.partner_id.id,
2816                     "partner_category_id": line.order_id.partner_id.category_id.ids,
2817                     "company_id": line.company_id.id,
2818                 })
2819             line.analytic_distribution = distribution or line.
2820             analytic_distribution
2821
2822     @api.depends('product_id', 'state', 'qty_invoiced', 'qty_delivered')
2823     def _compute_product_updatable(self):

```

```

2808         for line in self:
2809             if line.state == 'cancel':
2810                 line.product_updatable = False
2811             elif line.state == 'sale' and (
2812                 line.order_id.locked
2813                 or line.qty_invoiced > 0
2814                 or line.qty_delivered > 0
2815             ):
2816                 line.product_updatable = False
2817             else:
2818                 line.product_updatable = True
2819
2820     @api.depends('state')
2821     def _compute_product_uom_readonly(self):
2822         for line in self:
2823             # line.ids checks whether it's a new record not yet saved
2824             line.product_uom_readonly = line.ids and line.state in ['sale', 'cancel']
2825
2826     #=== CONSTRAINT METHODS ===#
2827
2828     #=== ONCHANGE METHODS ===#
2829
2830     @api.onchange('product_id')
2831     def _onchange_product_id_warning(self):
2832         if not self.product_id:
2833             return
2834
2835         product = self.product_id
2836         if product.sale_line_warn != 'no-message':
2837             if product.sale_line_warn == 'block':
2838                 self.product_id = False
2839
2840         return {
2841             'warning': {
2842                 'title': _("Warning for %s", product.name),
2843                 'message': product.sale_line_warn_msg,
2844             }
2845         }
2846
2847     @api.onchange('product_packaging_id')
2848     def _onchange_product_packaging_id(self):
2849         if self.product_packaging_id and self.product_uom_qty:
2850             newqty = self.product_packaging_id._check_qty(self.product_uom_qty, self.
2851                 product_uom, "UP")
2852             if float_compare(newqty, self.product_uom_qty, precision_rounding=self.
2853                 product_uom.rounding) != 0:
2854                 return {
2855                     'warning': {
2856                         'title': _('Warning'),
2857                         'message': _("
2858                             This product is packaged by %(pack_size).2f
2859                             %(pack_name)s. You should sell %(quantity).2f %(unit)s.",
2860                             pack_size=self.product_packaging_id.qty,
2861                             pack_name=self.product_id.uom_id.name,
2862                             quantity=newqty,
2863                             unit=self.product_uom.name
2864                         ),
2865                     },
2866                 }
2867
2868     #=== CRUD METHODS ===#
2869
2870     @api.model_create_multi
2871     def create(self, vals_list):
2872         for vals in vals_list:
2873             if vals.get('display_type') or self.default_get(['display_type']).get(
2874                 'display_type'):
2875                 vals['product_uom_qty'] = 0.0
2876
2877         lines = super().create(vals_list)
2878         if self.env.context.get('sale_no_log_for_new_lines'):
2879             return lines

```



```

2876
2877     for line in lines:
2878         if line.product_id and line.state == 'sale':
2879             msg = _("Extra line with %s", line.product_id.display_name)
2880             line.order_id.message_post(body=msg)
2881             # create an analytic account if at least an expense product
2882             if line.product_id.expense_policy not in [False, 'no'] and not line.
                order_id.analytic_account_id:
2883                 line.order_id._create_analytic_account()
2884
2885     return lines
2886
2887 def write(self, values):
2888     if 'display_type' in values and self.filtered(lambda line: line.display_type
        != values.get('display_type')):
2889         raise UserError(_("You cannot change the type of a sale order line.
            Instead you should delete the current line and create a new line of the
            proper type."))
2890
2891     if 'product_uom_qty' in values:
2892         precision = self.env['decimal.precision'].precision_get('Product Unit of
            Measure')
2893         self.filtered(
2894             lambda r: r.state == 'sale' and float_compare(r.product_uom_qty,
                values['product_uom_qty'], precision_digits=precision) != 0).
            _update_line_quantity(values)
2895
2896     # Prevent writing on a locked SO.
2897     protected_fields = self._get_protected_fields()
2898     if any(self.order_id.mapped('locked')) and any(f in values.keys() for f in
        protected_fields):
2899         protected_fields_modified = list(set(protected_fields) & set(values.keys
            ()))
2900
2901         if 'name' in protected_fields_modified and all(self.mapped(
            'is_downpayment')):
2902             protected_fields_modified.remove('name')
2903
2904         fields = self.env['ir.model.fields'].sudo().search([
2905             ('name', 'in', protected_fields_modified), ('model', '=', self._name)
2906         ])
2907         if fields:
2908             raise UserError(
2909                 _("It is forbidden to modify the following fields in a locked
                    order:\n%s",
                    '\n'.join(fields.mapped('field_description'))))
2910
2911
2912
2913     result = super().write(values)
2914
2915     # Don't recompute the package_id if we are setting the quantity of the items
    and the quantity of packages
2916     if 'product_uom_qty' in values and 'product_packaging_qty' in values and
        'product_packaging_id' not in values:
2917         self.env.remove_to_compute(self._fields['product_packaging_id'], self)
2918
2919     return result
2920
2921 def _get_protected_fields(self):
2922     """ Give the fields that should not be modified on a locked SO.
2923
2924     :returns: list of field names
2925     :rtype: list
2926     """
2927     return [
2928         'product_id', 'name', 'price_unit', 'product_uom', 'product_uom_qty',
2929         'tax_id', 'analytic_distribution'
2930     ]
2931
2932 def _update_line_quantity(self, values):
2933     orders = self.mapped('order_id')
2934     for order in orders:

```

```

2935         order_lines = self.filtered(lambda x: x.order_id == order)
2936         msg = Markup("<b>%s</b><ul>") % _("The ordered quantity has been updated."
)
2937         for line in order_lines:
2938             msg += Markup("<li> %s: <br/>") % line.product_id.display_name
2939             msg += _("
2940                 Ordered Quantity: %(old_qty)s -> %(new_qty)s",
2941                 old_qty=line.product_uom_qty,
2942                 new_qty=values["product_uom_qty"]
2943             ) + Markup("<br/>")
2944             if line.product_id.type in ('consu', 'product'):
2945                 msg += _("Delivered Quantity: %s", line.qty_delivered) + Markup(
2946                     "<br/>")
2947                 msg += _("Invoiced Quantity: %s", line.qty_invoiced) + Markup("<br/>")
2948             msg += Markup("</ul>")
2949         order.message_post(body=msg)
2950
2951     def _check_line_unlink(self):
2952         """ Check whether given lines can be deleted or not.
2953
2954         * Lines cannot be deleted if the order is confirmed.
2955         * Down payment lines who have not yet been invoiced bypass that exception.
2956         * Sections and Notes can always be deleted.
2957
2958         :returns: Sales Order Lines that cannot be deleted
2959         :rtype: `sale.order.line` recordset
2960         """
2961         return self.filtered(
2962             lambda line:
2963                 line.state == 'sale'
2964                 and (line.invoice_lines or not line.is_downpayment)
2965                 and not line.display_type
2966         )
2967
2968     @api.ondelete(at_uninstall=False)
2969     def _unlink_except_confirmed(self):
2970         if self._check_line_unlink():
2971             raise UserError(_("Once a sales order is confirmed, you can't remove one
2972                 of its lines (we need to track if something gets invoiced or
2973                 delivered).\n\
2974                 Set the quantity to 0 instead."))
2975
2976     #=== ACTION METHODS ===#
2977
2978     def action_add_from_catalog(self):
2979         order = self.env['sale.order'].browse(self.env.context.get('order_id'))
2980         return order.action_add_from_catalog()
2981
2982     #=== BUSINESS METHODS ===#
2983
2984     def _expected_date(self):
2985         self.ensure_one()
2986         if self.state == 'sale' and self.order_id.date_order:
2987             order_date = self.order_id.date_order
2988         else:
2989             order_date = fields.Datetime.now()
2990         return order_date + timedelta(days=self.customer_lead or 0.0)
2991
2992     def compute_uom_qty(self, new_qty, stock_move, rounding=True):
2993         return self.product_uom._compute_quantity(new_qty, stock_move.product_uom,
2994             rounding)
2995
2996     def _get_invoice_line_sequence(self, new=0, old=0):
2997         """
2998         Method intended to be overridden in third-party module if we want to prevent
2999         the resequencing
3000         of invoice lines.
3001
3002         :param int new:    the new line sequence
3003         :param int old:    the old line sequence
3004
3005         :return:           the sequence of the SO line, by default the new one.

```

```

3001         """
3002         return new or old
3003
3004     def _prepare_invoice_line(self, **optional_values):
3005         """Prepare the values to create the new invoice line for a sales order line.
3006
3007         :param optional_values: any parameter that should be added to the returned
3008         invoice line
3009         :rtype: dict
3010         """
3011         self.ensure_one()
3012         res = {
3013             'display_type': self.display_type or 'product',
3014             'sequence': self.sequence,
3015             'name': self.name,
3016             'product_id': self.product_id.id,
3017             'product_uom_id': self.product_uom.id,
3018             'quantity': self.qty_to_invoice,
3019             'discount': self.discount,
3020             'price_unit': self.price_unit,
3021             'tax_ids': [Command.set(self.tax_id.ids)],
3022             'sale_line_ids': [Command.link(self.id)],
3023             'is_downpayment': self.is_downpayment,
3024         }
3025         analytic_account_id = self.order_id.analytic_account_id.id
3026         if self.analytic_distribution and not self.display_type:
3027             res['analytic_distribution'] = self.analytic_distribution
3028         if analytic_account_id and not self.display_type:
3029             analytic_account_id = str(analytic_account_id)
3030             if 'analytic_distribution' in res:
3031                 res['analytic_distribution'][analytic_account_id] = res[
3032                     'analytic_distribution'].get(analytic_account_id, 0) + 100
3033             else:
3034                 res['analytic_distribution'] = {analytic_account_id: 100}
3035         if optional_values:
3036             res.update(optional_values)
3037         if self.display_type:
3038             res['account_id'] = False
3039         return res
3040
3041     def _prepare_procurement_values(self, group_id=False):
3042         """ Prepare specific key for moves or other components that will be created
3043         from a stock rule
3044         coming from a sale order line. This method could be override in order to add
3045         other custom key that could
3046         be used in move/po creation.
3047         """
3048         return {}
3049
3050     def _validate_analytic_distribution(self):
3051         for line in self.filtered(lambda l: not l.display_type and l.state in ['draft', 'sent']):
3052             line._validate_distribution(**{
3053                 'product': line.product_id.id,
3054                 'business_domain': 'sale_order',
3055                 'company_id': line.company_id.id,
3056             })
3057
3058     #=== CORE METHODS OVERRIDES ===#
3059
3060     def _get_partner_display(self):
3061         self.ensure_one()
3062         commercial_partner = self.order_partner_id.commercial_partner_id
3063         return f'({commercial_partner.ref or commercial_partner.name})'
3064
3065     def _additional_name_per_id(self):
3066         return {
3067             so_line.id: so_line._get_partner_display()
3068             for so_line in self
3069         }
3070
3071     #=== HOOKS ===#

```

```

3068
3069 def _is_delivery(self):
3070     self.ensure_one()
3071     return False
3072
3073 def _is_not_sellable_line(self):
3074     # True if the line is a computed line (reward, delivery, ...) that user
3075     # cannot add manually
3076     return False
3077
3078 def _get_product_catalog_lines_data(self, **kwargs):
3079     """ Return information about sale order lines in `self`.
3080
3081     If `self` is empty, this method returns only the default value(s) needed for
3082     the product
3083     catalog. In this case, the quantity that equals 0.
3084
3085     Otherwise, it returns a quantity and a price based on the product of the
3086     SOL(s) and whether
3087     the product is read-only or not.
3088
3089     A product is considered read-only if the order is considered read-only (see
3090     ``SaleOrder._is_readonly`` for more details) or if `self` contains multiple
3091     records.
3092
3093     Note: This method cannot be called with multiple records that have different
3094     products linked.
3095
3096     :raise odoo.exceptions.ValueError: ``len(self.product_id) != 1``
3097     :rtype: dict
3098     :return: A dict with the following structure:
3099         {
3100             'quantity': float,
3101             'price': float,
3102             'readOnly': bool,
3103         }
3104     """
3105     if len(self) == 1:
3106         return {
3107             'quantity': self.product_uom_qty,
3108             'price': self.price_unit,
3109             'readOnly': self.order_id._is_readonly(),
3110         }
3111     elif self:
3112         self.product_id.ensure_one()
3113         order_line = self[0]
3114         order = order_line.order_id
3115         return {
3116             'readOnly': True,
3117             'price': order.pricelist_id._get_product_price(
3118                 product=order_line.product_id,
3119                 quantity=1.0,
3120                 currency=order.currency_id,
3121                 date=order.date_order,
3122                 **kwargs,
3123             ),
3124             'quantity': sum(
3125                 self.mapped(
3126                     lambda line: line.product_uom._compute_quantity(
3127                         qty=line.product_uom_qty,
3128                         to_unit=line.product_id.uom_id,
3129                     )
3130                 )
3131             ),
3132         }
3133     else:
3134         return {
3135             'quantity': 0,
3136             # price will be computed in batch with pricelist utils so not given
3137             # here
3138         }

```

```

3134 #=== TOOLING ===#
3135
3136 def _convert_to_sol_currency(self, amount, currency):
3137     """Convert the given amount from the given currency to the SO(L) currency.
3138
3139     :param float amount: the amount to convert
3140     :param currency: currency in which the given amount is expressed
3141     :type currency: `res.currency` record
3142     :returns: converted amount
3143     :rtype: float
3144     """
3145     self.ensure_one()
3146     to_currency = self.currency_id or self.order_id.currency_id
3147     if currency and to_currency and currency != to_currency:
3148         conversion_date = self.order_id.date_order or fields.Date.context_today(
3149             self)
3150         company = self.company_id or self.order_id.company_id or self.env.company
3151         return currency._convert(
3152             from_amount=amount,
3153             to_currency=to_currency,
3154             company=company,
3155             date=conversion_date,
3156             round=False,
3157         )
3158     return amount
3159
3160 def has_valued_move_ids(self):
3161     return self.move_ids
3162
3163 # sale_report.py
3164 # -*- coding: utf-8 -*-
3165 # Part of Odoo. See LICENSE file for full copyright and licensing details.
3166
3167 from odoo import api, fields, models, tools
3168 from odoo.addons.sale.models.sale_order import SALE_ORDER_STATE
3169
3170
3171 class SaleReport(models.Model):
3172     _name = "sale.report"
3173     _description = "Sales Analysis Report"
3174     _auto = False
3175     _rec_name = 'date'
3176     _order = 'date desc'
3177
3178     @api.model
3179     def _get_done_states(self):
3180         return ['sale']
3181
3182     # sale.order fields
3183     name = fields.Char(string="Order Reference", readonly=True)
3184     date = fields.Datetime(string="Order Date", readonly=True)
3185     partner_id = fields.Many2one(comodel_name='res.partner', string="Customer",
3186                                 readonly=True)
3187     company_id = fields.Many2one(comodel_name='res.company', readonly=True)
3188     pricelist_id = fields.Many2one(comodel_name='product.pricelist', readonly=True)
3189     team_id = fields.Many2one(comodel_name='crm.team', string="Sales Team", readonly=
3190                               True)
3191     user_id = fields.Many2one(comodel_name='res.users', string="Salesperson", readonly=
3192                               True)
3193     state = fields.Selection(selection=SALE_ORDER_STATE, string="Status", readonly=
3194                               True)
3195     analytic_account_id = fields.Many2one(
3196         comodel_name='account.analytic.account', string="Analytic Account", readonly=
3197         True)
3198     invoice_status = fields.Selection(
3199         selection=[
3200             ('upselling', "Upselling Opportunity"),
3201             ('invoiced', "Fully Invoiced"),
3202             ('to invoice', "To Invoice"),
3203             ('no', "Nothing to Invoice"),
3204         ], string="Invoice Status", readonly=True)

```

```

3200
3201 campaign_id = fields.Many2one(comodel_name='utm.campaign', string="Campaign",
3202     readonly=True)
3203 medium_id = fields.Many2one(comodel_name='utm.medium', string="Medium", readonly=
3204     True)
3205 source_id = fields.Many2one(comodel_name='utm.source', string="Source", readonly=
3206     True)
3207
3208 # res.partner fields
3209 commercial_partner_id = fields.Many2one(
3210     comodel_name='res.partner', string="Customer Entity", readonly=True)
3211 country_id = fields.Many2one(
3212     comodel_name='res.country', string="Customer Country", readonly=True)
3213 industry_id = fields.Many2one(
3214     comodel_name='res.partner.industry', string="Customer Industry", readonly=True
3215     )
3216 partner_zip = fields.Char(string="Customer ZIP", readonly=True)
3217 state_id = fields.Many2one(comodel_name='res.country.state', string="Customer
3218     State", readonly=True)
3219
3220 # sale.order.line fields
3221 order_reference = fields.Reference(string='Related Order', selection=[(
3222     'sale.order', 'Sales Order')], group_operator="count_distinct")
3223
3224 categ_id = fields.Many2one(
3225     comodel_name='product.category', string="Product Category", readonly=True)
3226 product_id = fields.Many2one(
3227     comodel_name='product.product', string="Product Variant", readonly=True)
3228 product_tmpl_id = fields.Many2one(
3229     comodel_name='product.template', string="Product", readonly=True)
3230 product_uom = fields.Many2one(comodel_name='uom.uom', string="Unit of Measure",
3231     readonly=True)
3232 product_uom_qty = fields.Float(string="Qty Ordered", readonly=True)
3233 qty_to_deliver = fields.Float(string="Qty To Deliver", readonly=True)
3234 qty_delivered = fields.Float(string="Qty Delivered", readonly=True)
3235 qty_to_invoice = fields.Float(string="Qty To Invoice", readonly=True)
3236 qty_invoiced = fields.Float(string="Qty Invoiced", readonly=True)
3237 price_subtotal = fields.Monetary(string="Untaxed Total", readonly=True)
3238 price_total = fields.Monetary(string="Total", readonly=True)
3239 untaxed_amount_to_invoice = fields.Monetary(string="Untaxed Amount To Invoice",
3240     readonly=True)
3241 untaxed_amount_invoiced = fields.Monetary(string="Untaxed Amount Invoiced",
3242     readonly=True)
3243
3244 weight = fields.Float(string="Gross Weight", readonly=True)
3245 volume = fields.Float(string="Volume", readonly=True)
3246
3247 discount = fields.Float(string="Discount %", readonly=True)
3248 discount_amount = fields.Monetary(string="Discount Amount", readonly=True)
3249
3250 # aggregates or computed fields
3251 nbr = fields.Integer(string="# of Lines", readonly=True)
3252 currency_id = fields.Many2one(comodel_name='res.currency', compute=
3253     '_compute_currency_id')
3254
3255 @api.depends_context('allowed_company_ids')
3256 def _compute_currency_id(self):
3257     self.currency_id = self.env.company.currency_id
3258
3259 def _with_sale(self):
3260     return ""
3261
3262 def _select_sale(self):
3263     select_ = f"""
3264         MIN(l.id) AS id,
3265         l.product_id AS product_id,
3266         t.uom_id AS product_uom,
3267         CASE WHEN l.product_id IS NOT NULL THEN SUM(l.product_uom_qty / u.factor
3268             * u2.factor) ELSE 0 END AS product_uom_qty,
3269         CASE WHEN l.product_id IS NOT NULL THEN SUM(l.qty_delivered / u.factor *
3270             u2.factor) ELSE 0 END AS qty_delivered,
3271         CASE WHEN l.product_id IS NOT NULL THEN SUM((l.product_uom_qty -

```

```

3260         l.qty_delivered) / u.factor * u2.factor) ELSE 0 END AS qty_to_deliver,
CASE WHEN l.product_id IS NOT NULL THEN SUM(l.qty_invoiced / u.factor *
3261         u2.factor) ELSE 0 END AS qty_invoiced,
CASE WHEN l.product_id IS NOT NULL THEN SUM(l.qty_to_invoice / u.factor *
3262         u2.factor) ELSE 0 END AS qty_to_invoice,
CASE WHEN l.product_id IS NOT NULL THEN SUM(l.price_total
3263         / {self._case_value_or_one('s.currency_rate')})
3264         * {self._case_value_or_one('currency_table.rate')})
3265         ) ELSE 0
3266     END AS price_total,
CASE WHEN l.product_id IS NOT NULL THEN SUM(l.price_subtotal
3267         / {self._case_value_or_one('s.currency_rate')})
3268         * {self._case_value_or_one('currency_table.rate')})
3269         ) ELSE 0
3270     END AS price_subtotal,
CASE WHEN l.product_id IS NOT NULL THEN SUM(l.untaxed_amount_to_invoice
3271         / {self._case_value_or_one('s.currency_rate')})
3272         * {self._case_value_or_one('currency_table.rate')})
3273         ) ELSE 0
3274     END AS untaxed_amount_to_invoice,
CASE WHEN l.product_id IS NOT NULL THEN SUM(l.untaxed_amount_invoiced
3275         / {self._case_value_or_one('s.currency_rate')})
3276         * {self._case_value_or_one('currency_table.rate')})
3277         ) ELSE 0
3278     END AS untaxed_amount_invoiced,
COUNT(*) AS nbr,
3281     s.name AS name,
3282     s.date_order AS date,
3283     s.state AS state,
3284     s.invoice_status AS invoice_status,
3285     s.partner_id AS partner_id,
3286     s.user_id AS user_id,
3287     s.company_id AS company_id,
3288     s.campaign_id AS campaign_id,
3289     s.medium_id AS medium_id,
3290     s.source_id AS source_id,
3291     t.categ_id AS categ_id,
3292     s.pricelist_id AS pricelist_id,
3293     s.analytic_account_id AS analytic_account_id,
3294     s.team_id AS team_id,
3295     p.product_tmpl_id,
3296     partner.commercial_partner_id AS commercial_partner_id,
3297     partner.country_id AS country_id,
3298     partner.industry_id AS industry_id,
3299     partner.state_id AS state_id,
3300     partner.zip AS partner_zip,
3301     CASE WHEN l.product_id IS NOT NULL THEN SUM(p.weight * l.product_uom_qty
3302         / u.factor * u2.factor) ELSE 0 END AS weight,
3303     CASE WHEN l.product_id IS NOT NULL THEN SUM(p.volume * l.product_uom_qty
3304         / u.factor * u2.factor) ELSE 0 END AS volume,
3305     l.discount AS discount,
3306     CASE WHEN l.product_id IS NOT NULL THEN SUM(l.price_unit *
l.product_uom_qty * l.discount / 100.0
3307         / {self._case_value_or_one('s.currency_rate')})
3308         * {self._case_value_or_one('currency_table.rate')})
3309         ) ELSE 0
3310     END AS discount_amount,
concat('sale.order', ',', s.id) AS order_reference"""
3311
3312
3313     additional_fields_info = self._select_additional_fields()
3314     template = """
3315         %s AS %s"""
3316     for fname, query_info in additional_fields_info.items():
3317         select_ += template % (query_info, fname)
3318
3319     return select_
3320
3321     def _case_value_or_one(self, value):
3322         return f"""CASE COALESCE({value}, 0) WHEN 0 THEN 1.0 ELSE {value} END"""
3323
3324     def _select_additional_fields(self):
3325         """Hook to return additional fields SQL specification for select part of the

```

```

3326         table query.
3327
3328         :returns: mapping field -> SQL computation of field, will be converted to '_
3329         AS _field' in the final table definition
3330         :rtype: dict
3331         """
3332         return {}
3333
3334     def _from_sale(self):
3335         return """
3336         sale_order_line l
3337         LEFT JOIN sale_order s ON s.id=l.order_id
3338         JOIN res_partner partner ON s.partner_id = partner.id
3339         LEFT JOIN product_product p ON l.product_id=p.id
3340         LEFT JOIN product_template t ON p.product_tmpl_id=t.id
3341         LEFT JOIN uom_uom u ON u.id=l.product_uom
3342         LEFT JOIN uom_uom u2 ON u2.id=t.uom_id
3343         JOIN {currency_table} ON currency_table.company_id = s.company_id
3344         """.format(
3345             currency_table=self.env['res.currency']._get_query_currency_table(self.env
3346             .companies.ids, fields.Date.today())
3347         )
3348
3349     def _where_sale(self):
3350         return """
3351         l.display_type IS NULL"""
3352
3353     def _group_by_sale(self):
3354         return """
3355         l.product_id,
3356         l.order_id,
3357         t.uom_id,
3358         t.categ_id,
3359         s.name,
3360         s.date_order,
3361         s.partner_id,
3362         s.user_id,
3363         s.state,
3364         s.invoice_status,
3365         s.company_id,
3366         s.campaign_id,
3367         s.medium_id,
3368         s.source_id,
3369         s.pricelist_id,
3370         s.analytic_account_id,
3371         s.team_id,
3372         p.product_tmpl_id,
3373         partner.commercial_partner_id,
3374         partner.country_id,
3375         partner.industry_id,
3376         partner.state_id,
3377         partner.zip,
3378         l.discount,
3379         s.id,
3380         currency_table.rate"""
3381
3382     def _query(self):
3383         with_ = self._with_sale()
3384         return f"""
3385         {"WITH" + with_ + "(" if with_ else ""}
3386         SELECT {self._select_sale()}
3387         FROM {self._from_sale()}
3388         WHERE {self._where_sale()}
3389         GROUP BY {self._group_by_sale()}
3390         {"(" if with_ else ""}
3391         """
3392
3393     @property
3394     def _table_query(self):
3395         return self._query()
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499

```