GUÍA PRÁCTICA · POSTGRESQL · ODOO 16–18

SQL para Odoo

Consultas paso a paso con las tablas reales de Odoo
y casos de negocio simulados

Marlon Falcón Hernández

Implemento BIM 2.0 · Valencia, 2026

Índice

Introducción 3

Capítulo 1. Cómo guarda Odoo los datos 4

Capítulo 2. Conectarse y explorar la base de datos 6

Capítulo 3. Primeras consultas: los contactos 8

Capítulo 4. Unir tablas con JOIN 10

Capítulo 5. Ventas: totales, agrupaciones y rankings 12

Capítulo 6. Facturación y contabilidad 15

Capítulo 7. Compras 18

Capítulo 8. Inventario 20

Capítulo 9. CRM y proyectos 24

Capítulo 10. Técnicas avanzadas 27

Capítulo 11. Buenas prácticas y errores frecuentes 30

Anexo. Mapa de tablas y ejercicios 32

Introducción

Este libro enseña a sacar información de Odoo escribiendo consultas SQL directamente sobre su base de datos PostgreSQL. Todos los ejemplos usan las tablas reales de Odoo (versiones 16, 17 y 18) y se plantean como casos de negocio de una empresa ficticia, **Distribuciones Turia S.L.**, que vende, compra, almacena, factura y gestiona proyectos con Odoo.

A quién va dirigido. A personas del área de administración, finanzas, operaciones o sistemas que necesitan informes que Odoo no ofrece de serie, o que quieren entender de dónde salen los números. No hace falta saber programar: cada consulta se explica línea a línea.

Qué vas a aprender.

- Cómo se traduce un modelo de Odoo (por ejemplo, Pedidos de venta) a una tabla de base de datos.
- Escribir consultas de lectura: filtrar, ordenar, unir tablas, agrupar y calcular totales.
- Resolver preguntas reales: ventas por mes, deuda de clientes, stock por almacén, horas por proyecto.
- Técnicas avanzadas: subconsultas, CTE, funciones de ventana y vistas.

Qué necesitas.

1. Acceso a la base de datos PostgreSQL de Odoo con un **usuario de solo lectura** (capítulo 2).
2. Un cliente SQL: DBeaver, pgAdmin o la consola `psql`.
3. Idealmente, una **copia de la base de datos** (entorno de pruebas) para practicar sin tocar producción.

Aviso importante de seguridad. En este libro solo se usan consultas `SELECT`, que leen datos sin modificarlos. Nunca uses `INSERT`, `UPDATE` o `DELETE` sobre la base de datos de Odoo: saltarías las reglas de negocio, la contabilidad y los permisos, y podrías dejar datos incoherentes. Cualquier cambio de datos se hace desde la interfaz de Odoo o mediante su API.

Cómo leer este libro. Los capítulos 1 a 4 son la base y conviene leerlos en orden. Los capítulos 5 a 9 son casos por área (ventas, contabilidad, compras, inventario, CRM y proyectos) y pueden leerse sueltos. El capítulo 10 sube de nivel y el 11 recoge buenas prácticas. Al final hay un mapa de tablas y ejercicios con solución.

Capítulo 1. Cómo guarda Odoo los datos

Cada modelo de Odoo es una tabla de PostgreSQL con el mismo nombre, cambiando los puntos por guiones bajos. Conociendo esta regla y cinco tipos de campo, puedes leer casi cualquier dato de Odoo.

1.1 De modelo a tabla

Lo que ves en Odoo	Modelo técnico	Tabla SQL
Contactos (clientes, proveedores)	res.partner	res_partner
Usuarios	res.users	res_users
Pedidos de venta	sale.order	sale_order
Líneas de pedido de venta	sale.order.line	sale_order_line
Facturas y asientos	account.move	account_move
Apuntes contables	account.move.line	account_move_line
Productos (plantilla)	product.template	product_template
Productos (variante)	product.product	product_product

Para saber el modelo técnico de cualquier pantalla, activa el **modo desarrollador** (Ajustes → Activar modo desarrollador) y pasa el ratón por encima de un campo: verás el modelo y el nombre del campo.

1.2 Columnas que tienen todas las tablas

- `id`: identificador único y numérico de cada registro.
- `create_date` y `write_date`: fecha de creación y de última modificación (en UTC).
- `create_uid` y `write_uid`: usuario que creó y que modificó por última vez.
- `active` (en muchas tablas): si es `false`, el registro está **archivado** y Odoo lo oculta, pero sigue en la base de datos.
- `company_id` (en tablas multiempresa): a qué empresa pertenece el registro.

1.3 Tipos de relación

Many2one (muchos a uno). Es una columna que guarda el `id` de otro registro. Por ejemplo, `sale_order.partner_id` guarda el `id` del cliente en `res_partner`. Siempre termina en `_id`.

One2many (uno a muchos). No es una columna: es la otra cara de un many2one. Las líneas de un pedido no se guardan en `sale_order`, sino en `sale_order_line`, cuya columna `order_id` apunta al pedido.

Many2many (muchos a muchos). Se guarda en una tabla intermedia, normalmente terminada en `_rel`, con dos columnas de `id`. Por ejemplo, las etiquetas de contactos están en `res_partner_res_partner_category_rel` (`partner_id`, `category_id`) y los impuestos de cada línea de venta en `account_tax_sale_order_line_rel`.

1.4 Campos traducibles en formato JSON

Desde Odoo 16, los campos que se pueden traducir (nombre de producto, de país, de etapa, de proyecto) se guardan en formato `jsonb`, con un valor por idioma:

```
{"en_US": "Cement 25 kg", "es_ES": "Cemento 25 kg"}
```

Para leer el texto en español con respaldo en inglés:

```
SELECT COALESCE(name->>'es_ES', name->>'en_US') AS nombre
FROM product_template;
```

El operador `->>` extrae el valor de una clave como texto. Lo usaremos en todo el libro.

1.5 Estados y valores técnicos

Los campos de selección guardan una clave técnica en inglés, no la etiqueta que ves en pantalla. Un pedido confirmado tiene `state = 'sale'`, no 'Pedido de venta'. Cada capítulo incluye una tabla con los valores que necesitas.

1.6 Los identificadores externos

La tabla `ir_model_data` guarda los identificadores externos (por ejemplo, `base.es` para España). Sirven para localizar registros estándar sin depender de su `id`, que cambia entre bases de datos:

```
SELECT res_id AS id_espana
FROM ir_model_data
WHERE module = 'base' AND name = 'es' AND model = 'res.country';
```

Capítulo 2. Conectarse y explorar la base de datos

Antes de consultar, crea un usuario que solo pueda leer y aprende a encontrar tablas y columnas por tu cuenta. Con eso no dependes de nadie para descubrir dónde está un dato.

2.1 Crear un usuario de solo lectura

Lo hace el administrador del servidor una sola vez. Sustituye `odoo_prod` por el nombre de tu base de datos:

```
CREATE ROLE informes LOGIN PASSWORD 'cambia_esta_clave';
GRANT CONNECT ON DATABASE odoo_prod TO informes;
GRANT USAGE ON SCHEMA public TO informes;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO informes;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO informes;
```

La última línea hace que las tablas nuevas (al instalar módulos) también sean legibles. En Odoo.sh u Odoo Online el acceso directo a la base de datos está limitado; consulta con tu partner las opciones disponibles.

2.2 Herramientas para conectarse

Herramienta	Para quién	Ventaja
DBeaver	Usuarios de negocio	Gratuita, visual, exporta a Excel
pgAdmin	Administradores	Herramienta oficial de PostgreSQL
psql	Técnicos	Consola rápida, disponible en el servidor

Los datos de conexión son: servidor, puerto (normalmente 5432), nombre de la base de datos, usuario y contraseña.

2.3 Listar tablas y columnas

En `psql`, `\dt sale*` lista las tablas que empiezan por sale y `\d sale_order` muestra sus columnas. Con SQL estándar, válido en cualquier herramienta:

```sql
-- Tablas que contienen la palabra order
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public' AND table_name LIKE '%order%'
ORDER BY table_name;

-- Columnas de la tabla sale_order
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'sale_order'
ORDER BY ordinal_position;
```

2.4 Preguntarle a Odoo dónde está un campo

Odoo describe sus propios modelos y campos en las tablas `ir_model` e `ir_model_fields`. Esta consulta muestra todos los campos del modelo de pedidos de venta, con su tipo y el modelo al que apuntan:

```sql
SELECT f.name AS campo,
       f.ttype AS tipo,
       f.relation AS apunta_a,
       f.store AS guardado_en_tabla
FROM ir_model_fields f
WHERE f.model = 'sale.order'
ORDER BY f.name;
```

La columna `store` es clave: si vale `false`, el campo se calcula al vuelo y **no existe en la tabla**. Para esos campos hay que reconstruir el cálculo con SQL o leerlo desde la interfaz.

2.5 Primer vistazo seguro

Antes de lanzar una consulta sobre una tabla grande, añade siempre `LIMIT`:

```sql
SELECT * FROM sale_order LIMIT 10;
```

Capítulo 3. Primeras consultas: los contactos

Toda consulta de lectura sigue el mismo esqueleto: qué columnas (`SELECT`), de qué tabla (`FROM`), con qué condiciones (`WHERE`) y en qué orden (`ORDER BY`). Lo practicamos con `res_partner`, la tabla de clientes, proveedores y contactos.

3.1 Columnas útiles de res_partner

Columna	Significado
name	Nombre del contacto o empresa
is_company	true si es empresa, false si es persona
parent_id	Empresa a la que pertenece una persona de contacto
vat	NIF / CIF
email, phone	Correo y teléfono (en Odoo 17 o anterior también mobile)
street, zip, city	Dirección
state_id, country_id	Provincia y país (apuntan a res_country_state y res_country)
customer_rank, supplier_rank	Mayor que 0 si ha actuado como cliente o proveedor
active	false si está archivado

3.2 Caso: listado de clientes de Valencia

Distribuciones Turia quiere una lista de empresas clientes en la ciudad de Valencia, ordenada alfabéticamente.

```sql
SELECT name, vat, email, phone
FROM res_partner
WHERE is_company = true
  AND customer_rank > 0
  AND city ILIKE 'valencia'
  AND active = true
ORDER BY name;
```

Línea a línea:

- `SELECT name, vat, email, phone`: elige solo las columnas que necesitas. Evita `SELECT *` en informes.
- `WHERE ... AND ...`: todas las condiciones deben cumplirse a la vez.
- `ILIKE`: compara texto sin distinguir mayúsculas; encuentra Valencia, VALENCIA y valencia.
- `active = true`: excluye contactos archivados.

3.3 Buscar texto parcial

El comodín `%` representa cualquier texto. Para encontrar todos los contactos con un correo de Gmail:

```sql
SELECT name, email
FROM res_partner
WHERE email ILIKE '%@gmail.com';
```

3.4 Valores vacíos: NULL

Un campo vacío en Odoo es `NULL` en la base de datos. `NULL` no se compara con `=`, sino con `IS NULL` o `IS NOT NULL`.

Caso: clientes empresa sin NIF, para completar sus fichas antes de facturar.

```sql
SELECT id, name, city
FROM res_partner
WHERE is_company = true
  AND customer_rank > 0
  AND (vat IS NULL OR vat = '')
ORDER BY name;
```

Los paréntesis son importantes: sin ellos, el `OR` se aplicaría a toda la condición.

3.5 Empresas y sus personas de contacto

Una persona de contacto tiene `parent_id` apuntando a su empresa. Para ver los contactos de una empresa concreta:

```sql
SELECT name, function, email
FROM res_partner
WHERE parent_id = 42   -- id de la empresa
  AND active = true;
```

En el capítulo 4 haremos esto mismo mostrando el nombre de la empresa en lugar de su número.

3.6 Limitar y paginar

`LIMIT 20` devuelve las primeras 20 filas; `OFFSET 20` salta las 20 primeras. Combinado con `ORDER BY`, sirve para ver los últimos registros creados:

```sql
SELECT name, create_date
FROM res_partner
ORDER BY create_date DESC
LIMIT 20;
```

Capítulo 4. Unir tablas con JOIN

Un `JOIN` sustituye un número de `id` por el dato al que apunta: el nombre del cliente en lugar de su `partner_id`. Es la operación más usada con Odoo, porque casi toda la información está repartida entre varias tablas.

4.1 La idea

```
SELECT columnas
FROM tabla_a a
JOIN tabla_b b ON b.id = a.campo_id;
```

Las letras `a` y `b` son **alias**: nombres cortos para no repetir el nombre completo de la tabla. La condición `ON` dice qué columna de una tabla coincide con cuál de la otra.

4.2 Caso: contactos con su empresa

```
SELECT c.name   AS contacto,
       e.name   AS empresa,
       c.email
FROM res_partner c
JOIN res_partner e ON e.id = c.parent_id
WHERE c.active = true
ORDER BY e.name, c.name;
```

Aquí la misma tabla aparece dos veces con alias distintos: `c` para el contacto y `e` para la empresa.

4.3 INNER JOIN y LEFT JOIN

Tipo	Qué devuelve	Úsalo cuando
JOIN (o INNER JOIN)	Solo filas con pareja en ambas tablas	El dato relacionado es obligatorio
LEFT JOIN	Todas las filas de la izquierda; NULL si no hay pareja	El dato relacionado puede faltar

Caso: clientes con su provincia y país. Algunos clientes no tienen provincia, y con un `JOIN` normal desaparecerían del listado.

```
SELECT p.name AS cliente,
       p.city AS ciudad,
       st.name AS provincia,
       COALESCE(co.name->>'es_ES', co.name->>'en_US') AS pais
FROM res_partner p
LEFT JOIN res_country_state st ON st.id = p.state_id
LEFT JOIN res_country co ON co.id = p.country_id
WHERE p.customer_rank > 0
ORDER BY pais, provincia, cliente;
```

Recuerda del capítulo 1 que el nombre del país es un campo traducible en `jsonb`.

4.4 Caso: pedidos con cliente y comercial

El comercial de un pedido es un usuario (`res_users`), pero el nombre del usuario se guarda en su contacto (`res_partner`). Por eso se necesitan dos saltos:

```
SELECT so.name        AS pedido,
       so.date_order  AS fecha,
       cli.name       AS cliente,
       com.name       AS comercial,
       so.amount_total AS total
FROM sale_order so
JOIN res_partner cli     ON cli.id = so.partner_id
LEFT JOIN res_users u    ON u.id = so.user_id
LEFT JOIN res_partner com ON com.id = u.partner_id
WHERE so.state = 'sale'
ORDER BY so.date_order DESC;
```

El comercial va con `LEFT JOIN` porque un pedido puede no tener comercial asignado.

4.5 Unir a través de una tabla many2many

Caso: clientes con la etiqueta Mayorista. Las etiquetas viven en `res_partner_category` y la relación en una tabla intermedia:

```
SELECT p.name AS cliente
FROM res_partner p
JOIN res_partner_res_partner_category_rel rel ON rel.partner_id = p.id
JOIN res_partner_category cat ON cat.id = rel.category_id
WHERE COALESCE(cat.name->>'es_ES', cat.name->>'en_US') = 'Mayorista'
ORDER BY p.name;
```

4.6 Error típico: filas duplicadas

Si unes un pedido con sus líneas, cada pedido aparece una vez por línea. Si después sumas `amount_total` del pedido, el total sale multiplicado. Regla práctica: **suma importes al mismo nivel que la tabla de la que salen** (totales de pedido desde `sale_order`, importes de línea desde `sale_order_line`).

Capítulo 5. Ventas: totales, agrupaciones y rankings

`GROUP BY` convierte miles de filas en un resumen: una fila por mes, por cliente o por producto, con sumas y recuentos. Es la base de cualquier informe comercial.

5.1 Tablas y valores de ventas

Tabla	Columnas clave
sale_order	name, partner_id, user_id, team_id, date_order, state, amount_untaxed, amount_tax, amount_total, currency_id, company_id
sale_order_line	order_id, product_id, name, product_uom_qty, qty_delivered, qty_invoiced, price_unit, discount, price_subtotal, price_total, display_type
product_product	id, product_tmpl_id, default_code (referencia interna)
product_template	name (jsonb), categ_id, list_price, type

state en sale_order	Significado
draft	Presupuesto
sent	Presupuesto enviado
sale	Pedido confirmado
done	Bloqueado (solo hasta Odoo 16; desde la 17 se usa la columna locked)
cancel	Cancelado

En `sale_order_line`, las filas con `display_type` distinto de `NULL` son secciones y notas, no productos: exclúyelas siempre.

5.2 Funciones de agregación

Función	Calcula
COUNT(*)	Número de filas
SUM(x)	Suma
AVG(x)	Media
MIN(x), MAX(x)	Mínimo y máximo

5.3 Caso: ventas por mes

Dirección quiere la facturación comprometida (pedidos confirmados) por mes de 2026, sin impuestos.

```sql
SELECT date_trunc('month', so.date_order)::date AS mes,
       COUNT(*)                 AS pedidos,
       SUM(so.amount_untaxed)   AS base_imponible,
       ROUND(AVG(so.amount_untaxed), 2) AS ticket_medio
FROM sale_order so
WHERE so.state IN ('sale', 'done')
  AND so.date_order >= '2026-01-01'
  AND so.date_order <  '2027-01-01'
GROUP BY 1
ORDER BY 1;
```

`date_trunc('month', ...)` lleva cada fecha al primer día de su mes, y así todos los pedidos de un mes caen en el mismo grupo. `GROUP BY 1` agrupa por la primera columna del `SELECT`. Resultado simulado:

mes	pedidos	base_imponible	ticket_medio
2026-01-01	84	126.340,00	1.504,05
2026-02-01	91	139.812,50	1.536,40
2026-03-01	103	158.207,00	1.535,99

Ojo con las fechas: Odoo guarda `date_order` en **UTC**. Para agrupar por hora local de España usa `date_order AT TIME ZONE 'UTC' AT TIME ZONE 'Europe/Madrid'`.

5.4 Caso: los 10 mejores clientes

```sql
SELECT p.name AS cliente,
       COUNT(so.id) AS pedidos,
       SUM(so.amount_untaxed) AS ventas
FROM sale_order so
JOIN res_partner p ON p.id = so.partner_id
WHERE so.state IN ('sale', 'done')
  AND so.date_order >= '2026-01-01'
GROUP BY p.name
ORDER BY ventas DESC
LIMIT 10;
```

Si un cliente tiene varias direcciones de entrega y se le facturan pedidos a cada una, agrupa por la empresa: `COALESCE(p.parent_id, p.id)`, o usa la columna `commercial_partner_id` de `res_partner`, que siempre apunta a la empresa principal.

5.5 Caso: ventas por comercial con HAVING

`WHERE` filtra filas **antes** de agrupar; `HAVING` filtra grupos **después**. *Caso: comerciales que superan 50.000 € este año.*

```sql
SELECT com.name AS comercial,
       SUM(so.amount_untaxed) AS ventas
FROM sale_order so
JOIN res_users u ON u.id = so.user_id
JOIN res_partner com ON com.id = u.partner_id
WHERE so.state IN ('sale', 'done')
  AND so.date_order >= '2026-01-01'
GROUP BY com.name
HAVING SUM(so.amount_untaxed) > 50000
ORDER BY ventas DESC;
```

5.6 Caso: productos más vendidos

Aquí trabajamos a nivel de línea y unimos variante y plantilla para obtener el nombre:

```sql
SELECT pp.default_code AS referencia,
       COALESCE(pt.name->>'es_ES', pt.name->>'en_US') AS producto,
       SUM(sol.product_uom_qty) AS unidades,
       SUM(sol.price_subtotal)  AS importe
FROM sale_order_line sol
JOIN sale_order so       ON so.id = sol.order_id
JOIN product_product pp  ON pp.id = sol.product_id
JOIN product_template pt ON pt.id = pp.product_tmpl_id
WHERE so.state IN ('sale', 'done')
  AND sol.display_type IS NULL
  AND so.date_order >= '2026-01-01'
GROUP BY pp.default_code, producto
ORDER BY importe DESC
LIMIT 20;
```

5.7 Caso: pendiente de entregar y de facturar

```sql
SELECT so.name AS pedido,
       p.name  AS cliente,
       COALESCE(pt.name->>'es_ES', pt.name->>'en_US') AS producto,
       sol.product_uom_qty AS pedido_uds,
       sol.qty_delivered   AS entregado,
       sol.qty_invoiced    AS facturado,
       sol.product_uom_qty - sol.qty_delivered AS pendiente_entrega
FROM sale_order_line sol
JOIN sale_order so       ON so.id = sol.order_id
JOIN res_partner p       ON p.id = so.partner_id
JOIN product_product pp  ON pp.id = sol.product_id
JOIN product_template pt ON pt.id = pp.product_tmpl_id
WHERE so.state = 'sale'
  AND sol.display_type IS NULL
  AND sol.qty_delivered < sol.product_uom_qty
ORDER BY so.date_order;
```

Capítulo 6. Facturación y contabilidad

En Odoo, facturas, abonos, facturas de proveedor y asientos manuales están en la misma tabla, `account_move`, y se distinguen por `move_type`. El detalle contable (debe, haber, cuenta, impuesto) está en `account_move_line`.

6.1 Valores que hay que conocer

move_type	Documento
out_invoice	Factura de cliente
out_refund	Abono (rectificativa) de cliente
in_invoice	Factura de proveedor
in_refund	Abono de proveedor
entry	Asiento contable

Columna	Valores
state	draft (borrador), posted (publicado), cancel (cancelado)
payment_state	not_paid, partial, in_payment, paid, reversed

Regla de oro: **en informes contables filtra siempre `state = 'posted'`**. Los borradores no tienen validez contable.

Columnas de importe en `account_move`: `amount_untaxed`, `amount_tax`, `amount_total` y `amount_residual` (pendiente de cobro o pago), en la moneda del documento. Las versiones terminadas en `_signed` están en la moneda de la empresa y llevan signo: negativo en abonos.

6.2 Caso: facturas de cliente pendientes de cobro

```sql
SELECT am.name             AS factura,
       p.name              AS cliente,
       am.invoice_date     AS fecha,
       am.invoice_date_due AS vencimiento,
       am.amount_total     AS total,
       am.amount_residual  AS pendiente,
       CURRENT_DATE - am.invoice_date_due AS dias_vencida
FROM account_move am
JOIN res_partner p ON p.id = am.partner_id
WHERE am.move_type = 'out_invoice'
  AND am.state = 'posted'
  AND am.payment_state IN ('not_paid', 'partial')
ORDER BY am.invoice_date_due;
```

Restar dos fechas en PostgreSQL devuelve el número de días. Un valor positivo en `dias_vencida` significa que ya ha vencido.

6.3 Caso: antigüedad de la deuda (aging)

Finanzas quiere la deuda de cada cliente repartida por tramos de vencimiento. `CASE` permite crear columnas condicionales, y `SUM(CASE ...)` suma solo lo que cumple cada condición:

```sql
SELECT p.name AS cliente,
  SUM(CASE WHEN CURRENT_DATE <= am.invoice_date_due
           THEN am.amount_residual_signed ELSE 0 END) AS no_vencido,
  SUM(CASE WHEN CURRENT_DATE - am.invoice_date_due BETWEEN 1 AND 30
           THEN am.amount_residual_signed ELSE 0 END) AS d1_30,
  SUM(CASE WHEN CURRENT_DATE - am.invoice_date_due BETWEEN 31 AND 60
           THEN am.amount_residual_signed ELSE 0 END) AS d31_60,
  SUM(CASE WHEN CURRENT_DATE - am.invoice_date_due BETWEEN 61 AND 90
           THEN am.amount_residual_signed ELSE 0 END) AS d61_90,
  SUM(CASE WHEN CURRENT_DATE - am.invoice_date_due > 90
           THEN am.amount_residual_signed ELSE 0 END) AS mas_90,
  SUM(am.amount_residual_signed) AS total
FROM account_move am
JOIN res_partner p ON p.id = am.commercial_partner_id
WHERE am.move_type IN ('out_invoice', 'out_refund')
  AND am.state = 'posted'
  AND am.amount_residual <> 0
GROUP BY p.name
ORDER BY total DESC;
```

Incluir los abonos (`out_refund`) con importe firmado hace que resten de la deuda. Resultado simulado:

cliente	no_vencido	d1_30	d31_60	d61_90	mas_90	total
Obras Albufera S.A.	12.400	3.150	0	0	0	15.550
Reformas Ruzafa S.L.	2.100	0	4.800	0	1.920	8.820

Este cálculo se basa en la cabecera de factura. Si usas plazos de pago con varios vencimientos, el aging exacto se hace sobre `account_move_line` (columnas `date_maturity` y `amount_residual` de las líneas de cuentas a cobrar).

6.4 Caso: IVA repercutido y soportado por trimestre

Las líneas de impuesto de un asiento tienen `tax_line_id` informado (el impuesto que generó la cuota) y `tax_base_amount` (la base). `balance` es debe menos haber.

```sql
SELECT date_trunc('quarter', aml.date)::date AS trimestre,
       tax.name AS impuesto,
       SUM(aml.tax_base_amount) AS base,
       SUM(-aml.balance)        AS cuota
FROM account_move_line aml
JOIN account_tax tax ON tax.id = aml.tax_line_id
WHERE aml.parent_state = 'posted'
  AND aml.date >= '2026-01-01'
GROUP BY 1, 2
ORDER BY 1, 2;
```

La cuota sale positiva para el IVA repercutido (haber) y negativa para el soportado (debe). `parent_state` copia el estado del asiento en la línea y evita un `JOIN`. En Odoo 17 y 18 `account_tax.name` es traducible: usa `tax.name->>'es_ES'` en lugar de `tax.name`.

Para la declaración oficial usa siempre los informes fiscales de Odoo (modelo 303). Esta consulta sirve para análisis y cuadres.

6.5 Caso: saldo por cuenta contable

```sql
SELECT acc.code AS cuenta,
       COALESCE(acc.name->>'es_ES', acc.name->>'en_US') AS nombre,
       SUM(aml.debit)   AS debe,
       SUM(aml.credit)  AS haber,
       SUM(aml.balance) AS saldo
FROM account_move_line aml
JOIN account_account acc ON acc.id = aml.account_id
WHERE aml.parent_state = 'posted'
  AND aml.date <= '2026-09-30'
  AND aml.company_id = 1
GROUP BY acc.code, nombre
ORDER BY acc.code;
```

Diferencias entre versiones: en Odoo 16 `account_account.name` es texto normal (quita el `->>`); en Odoo 18 el código de cuenta depende de la empresa y la columna `code` ya no existe en la tabla: se guarda en `code_store`, un `jsonb` por empresa (`acc.code_store->>'1'` para la empresa 1).

Capítulo 7. Compras

Las compras siguen la misma estructura que las ventas: cabecera en `purchase_order` y líneas en `purchase_order_line`. Todo lo aprendido en el capítulo 5 se aplica aquí cambiando de tabla.

7.1 Tablas y estados

Tabla	Columnas clave
purchase_order	name, partner_id (proveedor), user_id (comprador), date_order, date_approve, date_planned, state, receipt_status, invoice_status, amount_untaxed, amount_total
purchase_order_line	order_id, product_id, product_qty, qty_received, qty_invoiced, price_unit, price_subtotal, date_planned, display_type

state	Significado
draft	Solicitud de presupuesto
sent	Solicitud enviada
to approve	Pendiente de aprobación
purchase	Pedido de compra confirmado
done	Bloqueado
cancel	Cancelado

`receipt_status` (desde Odoo 16) indica si la mercancía se ha recibido: `pending`, `partial` o `full`.

7.2 Caso: compras por proveedor en el año

```sql
SELECT p.name AS proveedor,
       COUNT(po.id) AS pedidos,
       SUM(po.amount_untaxed) AS importe
FROM purchase_order po
JOIN res_partner p ON p.id = po.partner_id
WHERE po.state IN ('purchase', 'done')
  AND po.date_approve >= '2026-01-01'
GROUP BY p.name
ORDER BY importe DESC;
```

Se filtra por `date_approve` (fecha de confirmación) porque `date_order` es la fecha del presupuesto y puede ser anterior.

7.3 Caso: pedidos con mercancía pendiente de recibir

El almacén quiere saber qué líneas debían llegar ya y aún no han llegado.

```sql
SELECT po.name AS pedido,
       p.name  AS proveedor,
       COALESCE(pt.name->>'es_ES', pt.name->>'en_US') AS producto,
       pol.product_qty  AS pedido_uds,
       pol.qty_received AS recibido,
       pol.product_qty - pol.qty_received AS pendiente,
       pol.date_planned::date AS fecha_prevista
FROM purchase_order_line pol
JOIN purchase_order po   ON po.id = pol.order_id
JOIN res_partner p       ON p.id = po.partner_id
JOIN product_product pp  ON pp.id = pol.product_id
JOIN product_template pt ON pt.id = pp.product_tmpl_id
WHERE po.state = 'purchase'
  AND pol.display_type IS NULL
  AND pol.qty_received < pol.product_qty
  AND pol.date_planned < now()
ORDER BY pol.date_planned;
```

7.4 Caso: comparar precios de compra entre proveedores

Compras quiere ver, para cada producto, el precio medio, mínimo y máximo pagado a cada proveedor.

```sql
SELECT COALESCE(pt.name->>'es_ES', pt.name->>'en_US') AS producto,
       p.name AS proveedor,
       COUNT(*)                          AS compras,
       ROUND(AVG(pol.price_unit), 2)     AS precio_medio,
       MIN(pol.price_unit)               AS precio_min,
       MAX(pol.price_unit)               AS precio_max
FROM purchase_order_line pol
JOIN purchase_order po   ON po.id = pol.order_id
JOIN res_partner p       ON p.id = po.partner_id
JOIN product_product pp  ON pp.id = pol.product_id
JOIN product_template pt ON pt.id = pp.product_tmpl_id
WHERE po.state IN ('purchase', 'done')
  AND pol.display_type IS NULL
GROUP BY producto, p.name
ORDER BY producto, precio_medio;
```

Esta consulta usa el precio de cada línea sin tener en cuenta la moneda ni la unidad de medida. Si compras en varias monedas o unidades (cajas y unidades sueltas), filtra por `po.currency_id` y por `pol.product_uom` antes de comparar.

7.5 Caso: pedidos recibidos sin factura de proveedor

```sql
SELECT po.name, p.name AS proveedor, po.amount_untaxed
FROM purchase_order po
JOIN res_partner p ON p.id = po.partner_id
WHERE po.state IN ('purchase', 'done')
  AND po.receipt_status = 'full'
  AND po.invoice_status = 'to invoice'
ORDER BY po.date_approve;
```

Capítulo 8. Inventario

El stock disponible en cada momento está en `stock_quant`; la historia de entradas y salidas, en `stock_move`. Odoo trata todo como movimientos entre **ubicaciones**: recibir de un proveedor es mover de la ubicación Proveedores a tu almacén.

8.1 Tablas y conceptos

Tabla	Para qué sirve	Columnas clave
stock_location	Ubicaciones físicas y virtuales	complete_name, usage, warehouse_id, active
stock_warehouse	Almacenes	name, code, lot_stock_id (ubicación de stock principal)
stock_quant	Existencias actuales	product_id, location_id, lot_id, quantity, reserved_quantity
stock_move	Movimientos	product_id, location_id (origen), location_dest_id (destino), quantity, state, date, picking_id
stock_picking	Albaranes y transferencias	name, picking_type_id, partner_id, state, scheduled_date, date_done, origin

usage en stock_location	Significado
internal	Ubicación real de tu almacén
supplier	Proveedores (virtual)
customer	Clientes (virtual)
inventory	Ajustes de inventario (virtual)
production	Fabricación (virtual)
transit	Tránsito entre almacenes
view	Agrupadora, no contiene stock

La cantidad hecha de un movimiento está en `stock_move.quantity` desde Odoo 17; en Odoo 16 la columna es `quantity_done`.

8.2 Caso: stock actual por almacén

```
SELECT w.name AS almacen,
       pp.default_code AS referencia,
       COALESCE(pt.name->>'es_ES', pt.name->>'en_US') AS producto,
       SUM(q.quantity)          AS existencias,
       SUM(q.reserved_quantity) AS reservado,
       SUM(q.quantity - q.reserved_quantity) AS disponible
FROM stock_quant q
JOIN stock_location l    ON l.id = q.location_id
JOIN stock_warehouse w   ON w.id = l.warehouse_id
JOIN product_product pp  ON pp.id = q.product_id
JOIN product_template pt ON pt.id = pp.product_tmpl_id
WHERE l.usage = 'internal'
GROUP BY w.name, pp.default_code, producto
HAVING SUM(q.quantity) <> 0
ORDER BY w.name, producto;
```

El filtro `l.usage = 'internal'` es imprescindible: los quants de ubicaciones virtuales tienen cantidades negativas que descuadrarían el total.

8.3 Caso: historial de movimientos de un producto

Un cliente reclama un pedido y hay que reconstruir qué pasó con la referencia CEM-25.

```
SELECT m.date,
       pk.name            AS albaran,
       lo.complete_name   AS desde,
       ld.complete_name   AS hacia,
       m.quantity         AS cantidad,
       m.origin           AS documento_origen
FROM stock_move m
JOIN product_product pp ON pp.id = m.product_id
JOIN stock_location lo  ON lo.id = m.location_id
JOIN stock_location ld  ON ld.id = m.location_dest_id
LEFT JOIN stock_picking pk ON pk.id = m.picking_id
WHERE pp.default_code = 'CEM-25'
  AND m.state = 'done'
ORDER BY m.date;
```

8.4 Caso: entradas y salidas por mes

Un movimiento es una **entrada** si su destino es interno y su origen no; es una **salida** al revés. Con `CASE` se clasifica cada movimiento:

```sql
SELECT date_trunc('month', m.date)::date AS mes,
  SUM(CASE WHEN ld.usage = 'internal' AND lo.usage <> 'internal'
           THEN m.quantity ELSE 0 END) AS entradas,
  SUM(CASE WHEN lo.usage = 'internal' AND ld.usage <> 'internal'
           THEN m.quantity ELSE 0 END) AS salidas
FROM stock_move m
JOIN stock_location lo ON lo.id = m.location_id
JOIN stock_location ld ON ld.id = m.location_dest_id
WHERE m.state = 'done'
  AND m.product_id = 125
GROUP BY 1
ORDER BY 1;
```

8.5 Caso: productos sin movimiento en 180 días

Productos con stock que no se han movido en seis meses: candidatos a liquidar. `NOT EXISTS` comprueba que no haya ninguna fila que cumpla la subconsulta:

```sql
SELECT pp.default_code,
       COALESCE(pt.name->>'es_ES', pt.name->>'en_US') AS producto,
       SUM(q.quantity) AS existencias
FROM stock_quant q
JOIN stock_location l    ON l.id = q.location_id AND l.usage = 'internal'
JOIN product_product pp  ON pp.id = q.product_id
JOIN product_template pt ON pt.id = pp.product_tmpl_id
WHERE NOT EXISTS (
  SELECT 1 FROM stock_move m
  WHERE m.product_id = q.product_id
    AND m.state = 'done'
    AND m.date >= now() - interval '180 days'
)
GROUP BY pp.default_code, producto
HAVING SUM(q.quantity) > 0
ORDER BY existencias DESC;
```

8.6 Caso: valoración del inventario

Con valoración automática o manual (coste medio o FIFO), Odoo 16 a 18 guarda cada capa de valor en `stock_valuation_layer`. La suma de `quantity` y `value` por producto da las existencias valoradas:

```sql
SELECT COALESCE(pt.name->>'es_ES', pt.name->>'en_US') AS producto,
       SUM(svl.quantity) AS cantidad,
       SUM(svl.value)    AS valor,
       ROUND(SUM(svl.value) / NULLIF(SUM(svl.quantity), 0), 4) AS coste_unitario
FROM stock_valuation_layer svl
JOIN product_product pp  ON pp.id = svl.product_id
JOIN product_template pt ON pt.id = pp.product_tmpl_id
WHERE svl.company_id = 1
  AND svl.create_date < '2026-10-01'
GROUP BY producto
HAVING SUM(svl.quantity) <> 0
ORDER BY valor DESC;
```

`NULLIF(x, 0)` evita el error de dividir por cero devolviendo `NULL`. Filtrar por `create_date` permite obtener la valoración a una fecha pasada, por ejemplo al cierre de un trimestre.

Capítulo 9. CRM y proyectos

El CRM se consulta en `crm_lead` (iniciativas y oportunidades) y `crm_stage` (etapas). Los proyectos, en `project_project` y `project_task`, y las horas imputadas en `account_analytic_line`.

9.1 Tablas del CRM

Tabla	Columnas clave
crm_lead	name, type (lead / opportunity), partner_id, user_id, team_id, stage_id, expected_revenue, probability, active, lost_reason_id, create_date, date_closed
crm_stage	name (jsonb), sequence, is_won

Cómo se reconoce cada situación:

- **Ganada**: su etapa tiene `is_won = true`.
- **Perdida**: `active = false` y `probability = 0`.
- **Abierta**: `active = true` y la etapa no es ganada.

9.2 Caso: embudo de ventas abierto

```sql
SELECT COALESCE(s.name->>'es_ES', s.name->>'en_US') AS etapa,
       COUNT(l.id) AS oportunidades,
       SUM(l.expected_revenue) AS importe_esperado,
       SUM(l.expected_revenue * l.probability / 100) AS importe_ponderado
FROM crm_lead l
JOIN crm_stage s ON s.id = l.stage_id
WHERE l.type = 'opportunity'
  AND l.active = true
GROUP BY s.sequence, etapa
ORDER BY s.sequence;
```

Agrupar también por `s.sequence` permite ordenar las etapas como aparecen en el tablero Kanban. Resultado simulado:

etapa	oportunidades	importe_esperado	importe_ponderado
Nueva	38	412.000	41.200
Cualificada	21	265.500	79.650
Propuesta	12	188.000	112.800
Ganada	9	96.300	96.300

9.3 Caso: tasa de conversión por comercial

De las oportunidades cerradas este año, ¿qué porcentaje ganó cada comercial? `FILTER` cuenta solo las filas que cumplen la condición:

```sql
SELECT com.name AS comercial,
       COUNT(*) FILTER (WHERE s.is_won)            AS ganadas,
       COUNT(*) FILTER (WHERE NOT l.active)        AS perdidas,
       ROUND(100.0 * COUNT(*) FILTER (WHERE s.is_won)
             / NULLIF(COUNT(*), 0), 1)             AS tasa_exito_pct
FROM crm_lead l
JOIN crm_stage s    ON s.id = l.stage_id
JOIN res_users u    ON u.id = l.user_id
JOIN res_partner com ON com.id = u.partner_id
WHERE l.type = 'opportunity'
  AND l.date_closed >= '2026-01-01'
  AND (s.is_won OR (l.active = false AND l.probability = 0))
GROUP BY com.name
ORDER BY tasa_exito_pct DESC;
```

Se multiplica por `100.0` (con decimal) para que PostgreSQL no haga una división entera, que devolvería 0.

9.4 Caso: motivos de pérdida

```sql
SELECT COALESCE(r.name->>'es_ES', r.name->>'en_US', 'Sin motivo') AS motivo,
       COUNT(*) AS oportunidades,
       SUM(l.expected_revenue) AS importe_perdido
FROM crm_lead l
LEFT JOIN crm_lost_reason r ON r.id = l.lost_reason_id
WHERE l.active = false AND l.probability = 0
  AND l.date_closed >= '2026-01-01'
GROUP BY motivo
ORDER BY importe_perdido DESC;
```

9.5 Tablas de proyectos

Tabla	Columnas clave
project_project	name (jsonb), partner_id, user_id (responsable), active, date_start, date
project_task	name, project_id, stage_id, date_deadline, allocated_hours (planned_hours en Odoo 16), parent_id, active
project_task_user_rel	task_id, user_id (asignados a la tarea)
account_analytic_line	date, project_id, task_id, employee_id, unit_amount (horas), amount (coste)

9.6 Caso: horas imputadas por proyecto y empleado

```sql
SELECT COALESCE(pr.name->>'es_ES', pr.name->>'en_US') AS proyecto,
       e.name AS empleado,
       SUM(al.unit_amount) AS horas,
       SUM(-al.amount)     AS coste
FROM account_analytic_line al
JOIN project_project pr ON pr.id = al.project_id
JOIN hr_employee e      ON e.id = al.employee_id
WHERE al.date >= '2026-10-01' AND al.date < '2026-11-01'
GROUP BY proyecto, e.name
ORDER BY proyecto, horas DESC;
```

El coste de las horas se guarda en negativo en `amount`; por eso se cambia de signo.

9.7 Caso: tareas con horas consumidas por encima de lo previsto

Aquí se agrupan primero las horas por tarea en una subconsulta y luego se comparan con lo previsto:

```
SELECT COALESCE(pr.name->>'es_ES', pr.name->>'en_US') AS proyecto,
       t.name AS tarea,
       t.allocated_hours AS previstas,
       h.horas AS imputadas,
       ROUND((h.horas / NULLIF(t.allocated_hours, 0) * 100)::numeric, 0) AS consumo_pct
FROM project_task t
JOIN project_project pr ON pr.id = t.project_id
JOIN (
  SELECT task_id, SUM(unit_amount) AS horas
  FROM account_analytic_line
  WHERE task_id IS NOT NULL
  GROUP BY task_id
) h ON h.task_id = t.id
WHERE t.allocated_hours > 0
  AND h.horas > t.allocated_hours
ORDER BY consumo_pct DESC;
```

Capítulo 10. Técnicas avanzadas

Con CTE, funciones de ventana y vistas se resuelven informes que antes requerían exportar a Excel: acumulados, comparativas con el año anterior, rankings dentro de cada grupo y consultas reutilizables.

10.1 CTE: consultas por pasos con WITH

Una CTE es una consulta con nombre que se usa después como si fuera una tabla. Hace legibles las consultas largas.

Caso: ventas mensuales de 2026 comparadas con el mismo mes de 2025.

```
WITH ventas_mes AS (
  SELECT date_trunc('month', date_order)::date AS mes,
         SUM(amount_untaxed) AS ventas
  FROM sale_order
  WHERE state IN ('sale', 'done')
    AND date_order >= '2025-01-01'
  GROUP BY 1
)
SELECT actual.mes,
       actual.ventas                  AS ventas_2026,
       anterior.ventas                AS ventas_2025,
       ROUND(100.0 * (actual.ventas - anterior.ventas)
             / NULLIF(anterior.ventas, 0), 1) AS variacion_pct
FROM ventas_mes actual
LEFT JOIN ventas_mes anterior
       ON anterior.mes = actual.mes - interval '1 year'
WHERE actual.mes >= '2026-01-01'
ORDER BY actual.mes;
```

10.2 Funciones de ventana: OVER

Una función de ventana calcula sobre un grupo de filas **sin colapsarlas**, a diferencia de `GROUP BY`. Sirve para acumulados, rankings y comparar cada fila con la anterior.

Función	Uso típico
SUM(x) OVER (ORDER BY fecha)	Acumulado año hasta la fecha
RANK() OVER (PARTITION BY grupo ORDER BY x DESC)	Ranking dentro de cada grupo
LAG(x) OVER (ORDER BY fecha)	Valor de la fila anterior
x / SUM(x) OVER ()	Porcentaje sobre el total

Caso: ventas por mes con acumulado anual y peso sobre el total.

```sql
WITH ventas_mes AS (
  SELECT date_trunc('month', date_order)::date AS mes,
         SUM(amount_untaxed) AS ventas
  FROM sale_order
  WHERE state IN ('sale', 'done')
    AND date_order >= '2026-01-01'
  GROUP BY 1
)
SELECT mes,
       ventas,
       SUM(ventas) OVER (ORDER BY mes)             AS acumulado,
       ventas - LAG(ventas) OVER (ORDER BY mes)    AS dif_mes_anterior,
       ROUND(100.0 * ventas / SUM(ventas) OVER (), 1) AS peso_pct
FROM ventas_mes
ORDER BY mes;
```

Caso: los 3 productos más vendidos de cada categoría. `PARTITION BY` reinicia el ranking en cada categoría:

```sql
WITH ventas_producto AS (
  SELECT pc.complete_name AS categoria,
         COALESCE(pt.name->>'es_ES', pt.name->>'en_US') AS producto,
         SUM(sol.price_subtotal) AS importe
  FROM sale_order_line sol
  JOIN sale_order so        ON so.id = sol.order_id
  JOIN product_product pp   ON pp.id = sol.product_id
  JOIN product_template pt  ON pt.id = pp.product_tmpl_id
  JOIN product_category pc  ON pc.id = pt.categ_id
  WHERE so.state IN ('sale', 'done') AND sol.display_type IS NULL
  GROUP BY categoria, producto
),
ranking AS (
  SELECT *, RANK() OVER (PARTITION BY categoria ORDER BY importe DESC) AS puesto
  FROM ventas_producto
)
SELECT categoria, puesto, producto, importe
FROM ranking
WHERE puesto <= 3
ORDER BY categoria, puesto;
```

10.3 Subconsultas

Una subconsulta es un `SELECT` dentro de otro. Se usa en tres sitios:

- En `WHERE ... IN (SELECT ...)`: filtrar por una lista calculada.
- En `WHERE EXISTS` / `NOT EXISTS`: comprobar si existe algo relacionado (capítulo 8.5).
- En `FROM ( SELECT ... ) alias`: agregar primero y unir después (capítulo 9.7).

Caso: clientes que compraron en 2025 pero no en 2026.

```sql
SELECT p.name
FROM res_partner p
WHERE p.id IN (
        SELECT partner_id FROM sale_order
        WHERE state IN ('sale', 'done')
          AND date_order >= '2025-01-01' AND date_order < '2026-01-01')
  AND p.id NOT IN (
        SELECT partner_id FROM sale_order
        WHERE state IN ('sale', 'done')
          AND date_order >= '2026-01-01')
ORDER BY p.name;
```

10.4 Funciones útiles

Función	Qué hace	Ejemplo
COALESCE(a, b)	Primer valor no nulo	COALESCE(p.phone, 'sin teléfono')
NULLIF(a, b)	NULL si a = b	NULLIF(cantidad, 0)
ROUND(x, n)	Redondea a n decimales	ROUND(importe, 2)
date_trunc('month', f)	Inicio de mes, trimestre o año	date_trunc('quarter', fecha)
EXTRACT(YEAR FROM f)	Parte de una fecha	EXTRACT(MONTH FROM fecha)
to_char(f, 'YYYY-MM')	Fecha como texto	to_char(fecha, 'DD/MM/YYYY')
string_agg(x, ', ')	Une textos de un grupo	Lista de productos de un pedido
campo->>'es_ES'	Texto de un campo traducible	Nombre de producto en español

10.5 Vistas: guardar una consulta

Una vista guarda una consulta con nombre para reutilizarla en otras consultas o en Excel y Power BI. Créalas en un esquema propio, nunca en el esquema `public` de Odoo, para no interferir con las actualizaciones:

```sql
CREATE SCHEMA IF NOT EXISTS informes;

CREATE OR REPLACE VIEW informes.ventas_mensuales AS
SELECT date_trunc('month', date_order)::date AS mes,
       company_id,
       SUM(amount_untaxed) AS ventas
FROM public.sale_order
WHERE state IN ('sale', 'done')
GROUP BY 1, 2;

GRANT USAGE ON SCHEMA informes TO informes;
GRANT SELECT ON ALL TABLES IN SCHEMA informes TO informes;
```

Crear vistas no modifica datos de Odoo, pero sí requiere un usuario con permiso de creación. Tras actualizar Odoo de versión, revisa las vistas: si una columna cambia de nombre, la vista deja de funcionar.

Capítulo 11. Buenas prácticas y errores frecuentes

La mayoría de los números que no cuadran con Odoo se deben a cinco despistes: incluir borradores, mezclar empresas, mezclar monedas, olvidar los archivados o duplicar filas con un `JOIN`. Esta lista sirve para revisar cualquier consulta antes de entregar un informe.

11.1 Lista de comprobación

☐ ¿Filtro por estado válido? (`state = 'posted'` en contabilidad, `state IN ('sale','done')` en ventas).
☐ ¿Filtro por empresa? Con varias empresas, añade `company_id = X` o agrupa por `company_id`.
☐ ¿Todos los importes están en la misma moneda? Usa las columnas `_signed` o `balance` (moneda de la empresa) cuando hay documentos en divisa.
☐ ¿Debo incluir o excluir registros archivados (`active`)?
☐ ¿He excluido secciones y notas (`display_type IS NULL`) en líneas de pedido?
☐ ¿Algún `JOIN` multiplica filas antes de sumar?
☐ ¿Las fechas con hora están en UTC y debo convertirlas a hora de Madrid?
☐ ¿El resultado coincide con el informe equivalente de Odoo para un periodo de prueba?

11.2 Nunca escribir en la base de datos

Odoo calcula importes, saldos, stock y secuencias en su código Python, no en la base de datos. Un `UPDATE` directo no recalcula nada: puede dejar facturas con totales que no coinciden con sus líneas, stock que no coincide con los movimientos o asientos descuadrados. Cualquier corrección se hace desde la interfaz, con una importación o mediante la API (XML-RPC / JSON-RPC).

11.3 Rendimiento

- Filtra siempre por fecha en tablas grandes (`account_move_line`, `stock_move`, `mail_message`).
- Usa `LIMIT` mientras desarrollas la consulta.
- Lanza los informes pesados fuera del horario de trabajo o sobre una réplica de la base de datos.
- Antepón `EXPLAIN ANALYZE` a una consulta lenta para ver su plan de ejecución; si aparece Seq Scan sobre una tabla de millones de filas, falta un filtro o un índice.
- No crees índices en la base de producción sin hablar con tu partner de Odoo.

11.4 Campos que no están en la tabla

Algunos campos que ves en pantalla son calculados y no se guardan (`store = false` en `ir_model_fields`, capítulo 2.4). Ejemplos habituales: `qty_available` y `virtual_available` de productos, o `total_due` de contactos. En SQL se reconstruyen a partir de su origen: el stock desde `stock_quant`, la deuda desde `account_move_line`.

11.5 Diferencias entre versiones

Dato	Odoo 16	Odoo 17	Odoo 18
Cantidad hecha en stock_move	quantity_done	quantity	quantity
Horas previstas en project_task	planned_hours	allocated_hours	allocated_hours
Pedido de venta bloqueado	state = 'done'	columna locked	columna locked
Código de cuenta contable	code	code	code_store (jsonb por empresa)
Producto almacenable	type = 'product'	type = 'product'	type = 'consu' y is_storable = true
Nombre de cuenta e impuesto	texto	jsonb	jsonb

Antes de reutilizar una consulta en otra base de datos, comprueba su versión: `SELECT latest_version FROM ir_module_module WHERE name = 'base';`.

11.6 Privacidad

Las tablas de Odoo contienen datos personales (contactos, empleados, nóminas). Da acceso de solo lectura únicamente a quien lo necesite, evita exportar datos personales sin necesidad y cumple el RGPD al compartir informes.

Anexo. Mapa de tablas y ejercicios

A.1 Mapa rápido modelo → tabla

Área	Modelo	Tabla	Se une con
Base	res.partner	res_partner	parent_id, country_id, state_id, commercial_partner_id
Base	res.users	res_users	partner_id → res_partner
Base	res.company	res_company	partner_id, currency_id
Base	res.currency	res_currency	—
Productos	product.template	product_template	categ_id → product_category
Productos	product.product	product_product	product_tmpl_id → product_template
Ventas	sale.order	sale_order	partner_id, user_id, team_id
Ventas	sale.order.line	sale_order_line	order_id, product_id
Compras	purchase.order	purchase_order	partner_id, user_id
Compras	purchase.order.line	purchase_order_line	order_id, product_id
Contabilidad	account.move	account_move	partner_id, journal_id, commercial_partner_id
Contabilidad	account.move.line	account_move_line	move_id, account_id, partner_id, tax_line_id
Contabilidad	account.account	account_account	—
Contabilidad	account.journal	account_journal	—
Contabilidad	account.tax	account_tax	—
Inventario	stock.quant	stock_quant	product_id, location_id, lot_id
Inventario	stock.move	stock_move	product_id, location_id, location_dest_id, picking_id
Inventario	stock.picking	stock_picking	partner_id, picking_type_id
Inventario	stock.location	stock_location	warehouse_id
CRM	crm.lead	crm_lead	stage_id, user_id, team_id, partner_id
CRM	crm.stage	crm_stage	—
Proyectos	project.project	project_project	partner_id, user_id
Proyectos	project.task	project_task	project_id, stage_id
Proyectos	account.analytic.line	account_analytic_line	project_id, task_id, employee_id
RR. HH.	hr.employee	hr_employee	user_id, department_id

A.2 Ejercicios

Intenta resolverlos antes de mirar la solución.

Ejercicio 1. Lista los proveedores (`supplier_rank > 0`) de España sin correo electrónico.

```sql
SELECT p.name, p.phone
FROM res_partner p
JOIN res_country c ON c.id = p.country_id
WHERE p.supplier_rank > 0
  AND c.code = 'ES'
  AND (p.email IS NULL OR p.email = '')
  AND p.active = true
ORDER BY p.name;
```

Ejercicio 2. Número de presupuestos que nunca se confirmaron, por comercial, en 2026.

```sql
SELECT com.name AS comercial, COUNT(*) AS presupuestos_sin_confirmar
FROM sale_order so
JOIN res_users u     ON u.id = so.user_id
JOIN res_partner com ON com.id = u.partner_id
WHERE so.state IN ('draft', 'sent', 'cancel')
  AND so.create_date >= '2026-01-01'
GROUP BY com.name
ORDER BY 2 DESC;
```

Ejercicio 3. Importe facturado neto (facturas menos abonos) por cliente en 2026, en moneda de la empresa.

```sql
SELECT p.name AS cliente,
       SUM(am.amount_untaxed_signed) AS facturado_neto
FROM account_move am
JOIN res_partner p ON p.id = am.commercial_partner_id
WHERE am.move_type IN ('out_invoice', 'out_refund')
  AND am.state = 'posted'
  AND am.invoice_date >= '2026-01-01'
GROUP BY p.name
ORDER BY facturado_neto DESC;
```

Ejercicio 4. Albaranes de salida retrasados: pendientes de validar y con fecha prevista ya pasada.

```sql
SELECT pk.name, p.name AS cliente, pk.scheduled_date, pk.origin
FROM stock_picking pk
JOIN stock_picking_type t ON t.id = pk.picking_type_id
LEFT JOIN res_partner p   ON p.id = pk.partner_id
WHERE t.code = 'outgoing'
  AND pk.state NOT IN ('done', 'cancel')
  AND pk.scheduled_date < now()
ORDER BY pk.scheduled_date;
```

Ejercicio 5. Para cada cliente, la fecha de su último pedido y los días transcurridos desde entonces, ordenado de más a menos inactivo.

```sql
SELECT p.name AS cliente,
       MAX(so.date_order)::date AS ultimo_pedido,
       CURRENT_DATE - MAX(so.date_order)::date AS dias_sin_comprar
FROM sale_order so
JOIN res_partner p ON p.id = so.partner_id
WHERE so.state IN ('sale', 'done')
GROUP BY p.name
ORDER BY dias_sin_comprar DESC;
```

Ejercicio 6 (avanzado). El cliente principal de cada comercial, usando una función de ventana.

```sql
WITH ventas AS (
  SELECT so.user_id, so.partner_id, SUM(so.amount_untaxed) AS importe
  FROM sale_order so
  WHERE so.state IN ('sale', 'done') AND so.date_order >= '2026-01-01'
  GROUP BY so.user_id, so.partner_id
),
ranking AS (
  SELECT *, ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY importe DESC) AS n
  FROM ventas
)
SELECT com.name AS comercial, cli.name AS cliente_principal, r.importe
FROM ranking r
JOIN res_users u     ON u.id = r.user_id
JOIN res_partner com ON com.id = u.partner_id
JOIN res_partner cli ON cli.id = r.partner_id
WHERE r.n = 1
ORDER BY r.importe DESC;
```