

IMPLEMENTACIÓN DE UNA BASE DE DATOS PARA LA BODEGA JESSICA

Hito Base de Datos I - Grupo 5

Mayo 2022



Integrantes:
Joaquín Francisco Jordán O'Connor
Juan Diego Castro Padilla
Enzo Gabriel Camizan Vidal
José Rafael Chachi Rodriguez
Samanta Susana Chang Kuoman

Universidad de Ingeniería y Tecnología

Ciencia de la Computación

Docente: Chambilla Aquino, Teófilo

2022-1

Índice

1. Requisitos	4
1.1. Introducción	4
1.2. Descripción general del problema/organización/empresa	4
1.3. Necesidad/usos de la base de datos	4
1.4. ¿Cómo resuelve el problema hoy?	5
1.4.1. Cómo se almacenan/procesa los datos hoy	5
1.4.2. Flujo de datos	5
1.5. Descripción detallada del sistema	5
1.5.1. Objetos de información actuales	5
1.5.2. Características y funcionalidades esperadas	6
1.5.3. Tipos de usuarios existentes/necesarios	6
1.5.4. Tipos de consulta, actualizaciones	6
1.5.5. Tamaño estimado de la base de datos	7
1.6. Objetivos del proyecto	9
1.7. Referencias del proyecto	9
1.8. Eventualidades	9
1.8.1. Problemas que pudieran encontrarse en el proyecto.	9
1.8.2. Límites y alcances del proyecto	10
2. Modelo Entidad-Relación	10
2.1. Reglas semánticas	10
2.2. Modelo Entidad-Relación	12
2.3. Especificaciones y consideraciones sobre el modelo	12
2.3.1. Entidad Persona:	12
2.3.2. Entidad Trabajador:	13
2.3.3. Entidad ClienteDelivery:	13
2.3.4. Entidad ClienteLocal	13
2.3.5. Entidad Turno:	13
2.3.6. Entidad Repartidor:	14
2.3.7. Entidad TrabajadorLocal:	14
2.3.8. Entidad Local:	14
2.3.9. Entidad Caja:	14
2.3.10. Entidad Producto:	15
2.3.11. Entidad Empresa:	15
2.3.12. Entidad Compra:	15
2.3.13. Entidad Delivery:	15
2.3.14. Entidad CompraLocal:	16
2.3.15. Entidad EntregaProductos:	16
3. Modelo Relacional	16
3.1. Modelo Relacional	16
3.2. Especificaciones de transformación	17
3.2.1. Entidades	17
3.2.2. Entidades débiles	17

3.2.3. Entidades superclase/subclase	18
3.2.4. Relaciones binarias	19
3.2.5. Relaciones ternarias	19
3.3. Diccionario de datos	19
4. Implementación de la base de datos	25
4.1. Creación de tablas en PostgreSQL	25
4.2. Carga de datos	33
4.3. Simulación de datos faltantes	34
5. Optimización y experimentación	34
5.1. Consultas SQL para el experimento	34
5.1.1. Descripción del tipo de consultas seleccionadas	34
5.1.2. Implementación de consultas en SQL	35
5.2. Metodología del experimento	37
5.2.1. Prueba sin índices	37
5.2.2. Prueba con índices	37
5.3. Optimización de consultas	38
5.3.1. Planes de índices para Consulta 1	38
5.3.2. Planes de índices para Consulta 2	48
5.3.3. Planes de índices para Consulta 3	73
5.4. Plataforma de pruebas	97
5.5. Medición de tiempos	98
5.5.1. Sin índices	98
5.5.2. Con índices	99
5.6. Resultados	100
5.6.1. Consulta 1	100
5.6.2. Consulta 2	101
5.6.3. Consulta 3	102
5.7. Análisis y discusión	102
6. Conclusiones	102
7. Anexos	103
7.1. Normalización de la base de datos	103
7.2. Dump Data	106
7.3. Modelo Físico	107
7.4. Videos de experimentación	107
7.4.1. Consulta 1	107
7.4.2. Consulta 2	107
7.4.3. Consulta 3	107
7.5. Frontend para la base de datos	107
7.6. Pregunta extra	108

1. Requisitos

1.1. Introducción

La bodega Jessica es una bodega familiar que abrió en el año 1980 durante los inicios de Villa El Salvador. Surgió como una necesidad para generar ingresos en la familia Rodriguez Espinoza a causa de la migración interna de la época. A través de los años, el negocio se ha ido expandiendo, llegando a generar más locales hasta lograr a ser una bodega reconocida por todo el barrio. El reconocimiento era tan grande que casi todos de la avenida en donde se ubicaba el local principal conocían al dueño.

Lamentablemente, en la década de 2010, el aumento de la delincuencia perjudicó a la mayoría de los locales en la capital, la bodega Jessica no fue la excepción. A pesar de 3 robos realizados a la empresa, esta continuó atendiendo a sus clientes como todos los días.

A vísperas del año 2020, se deseaba realizar un gran proyecto en el que se implementaría un sistema de delivery, además de expandir aún más la variedad de productos ofrecidos y brindar otros servicios. No obstante, debido a la coyuntura actual, la pérdida de trabajadores y el hecho de registrar todo a lápiz y papel, se ha dificultado el manejo del negocio, generando muy pocos ingresos e inclusive cayendo en deudas.

El presente proyecto busca implementar un modelo de base de datos que facilite el registro de las ventas realizadas e inventario de la bodega en los distintos locales y el tráfico de datos con el fin de llevar adelante la bodega familiar haciendo de manera más cómoda y organizada las ventas.

1.2. Descripción general del problema/organización/empresa

Si bien es cierto que la empresa tiene años funcionando y ofreciendo su servicio a muchísimos clientes, desde el inicio del establecimiento siempre se ha optado por registrar las ventas y realizar inventario a lápiz y papel, todo esto a raíz que nunca existió la necesidad de implementar algún sistema de base de datos. Actualmente, la falta de personal obligó a la empresa a administrar de mejor manera las ventas realizadas, por lo cual se requiere una base de datos que pueda optimizar el registro de ventas y manejar la información almacenada de sus productos.

1.3. Necesidad/usos de la base de datos

La bodega Jessica requiere modernizar su sistema de almacenamiento de datos para poder expandirse. Si bien ahora cuentan con dos locales y no cuentan con un servicio de delivery, la implementación de estas mejoras requiere de una

base de datos robusta.

La bodega planea implementar un sistema de delivery por medio de WhatsApp, y contratará repartidores para llevar esto a cabo. Esto le permitirá destacar sobre muchas otras bodegas en Villa el Salvador que no cuentan con sistemas de delivery.

Por otro lado, la bodega planea abrir varios locales, para lo cuál requiere saber el stock de cada producto en cada local, así como los trabajadores que tiene cada local y las compras registradas en cada una de las cajas, lo cuál les permitirá determinar la rentabilidad de cada uno de sus locales de manera rápida y efectiva.

1.4. ¿Cómo resuelve el problema hoy?

1.4.1. Cómo se almacenan/procesa los datos hoy

Actualmente, la bodega registra todas las ventas realizadas por día en un cuaderno, donde se anota el producto y la cantidad vendida, además del monto. Se registran los ingresos semanales sumando el monto de cada día utilizando una calculadora. Para el proceso de reabastecimiento de la tienda, se anotan los pedidos realizados en otro cuaderno, pero antes de ello se visualiza el stock producto por producto.

1.4.2. Flujo de datos

Los datos que maneja la empresa principalmente provienen de la interacción que puede tener un producto, además de sus propias características. Las ventas y reabastecimiento de estos generan un flujo de datos que se recolecta información para, posteriormente, almacenarla y usarla.

1.5. Descripción detallada del sistema

1.5.1. Objetos de información actuales

La información más actual sobre la bodega Jessica se encuentra dispersa en cuadernos donde se escribe el nombre del producto y la cantidad vendida, esta ha sido la forma tradicional de registrar las ventas diarias. Los datos mencionados anteriormente serán insertados en la base de datos para su posterior utilización. Sin embargo, como parte del desarrollo del proyecto, se plantea realizar consultas considerando escenarios de 1 000 (mil) datos, 10 000 (diez mil) datos, 100 000 (cien mil) datos y 1 000 000 (un millón) de datos almacenados en la base de datos. Para ello, se elaborarán scripts que permitan la generación de datos aleatorios.

1.5.2. Características y funcionalidades esperadas

Se tiene la expectativa de poder diseñar e implementar una base de datos que garantice la integridad de los registros de la empresa. Posterior a su realización, se espera que la base de datos pueda almacenar y actualizar la información sobre ventas, stocks de productos y su abastecimiento por cada local. De esta manera, se reduciría increíblemente el esfuerzo de la empresa para obtener cuentas sobre ganancias, conocer los stocks de productos por local o incluso hacer un seguimiento de la efectividad del nuevo sistema de delivery que planean implementar.

1.5.3. Tipos de usuarios existentes/necesarios

A día de hoy, la bodega Jessica cuenta con 8 trabajadores de local, 4 potenciales repartidores, y un administrador de local por cada uno de los 2 locales con los que cuenta actualmente. Finalmente, el rol más importante en la organización de este negocio pertenece a la dueña de las bodegas. Poder definir estos roles es de vital importancia para determinar los niveles de acceso a la base de datos por cada usuario.

En el modelo, la base de datos contará con un usuario por cada trabajador. Este usuario tendrá acceso a los deliveries, los productos de la bodega, así como sus correspondientes stocks por local.

Por otra parte, los repartidores de delivery tendrán acceso de lectura para permitirles tener conocimiento de las entregas que tienen pendientes; además, podrán visualizar el ítem de compra del delivery para verificar que los productos a entregar son los correctos.

En cuanto al personal administrativo, se les brindará acceso a la información de todos los empleados (trabajadores y repartidores), proveedores, fechas de abastecimiento del local, así como el registro de todas las compras. Sin embargo, a pesar de tener acceso a toda la base de datos, el administrador no tiene el derecho de eliminar información de la misma, puesto que solo tiene el deber de realizar consultas sobre las ganancias, el rendimiento de los trabajadores, etcétera.

Por último, el dueño del negocio tendrá acceso sin limitaciones a la base de datos, este podrá agregar y quitar proveedores de productos, agregar productos nuevos en el mercado que sean de interés para el negocio, despedir y contratar empleados, y todo lo que considere necesario para la gestión de las bodegas.

1.5.4. Tipos de consulta, actualizaciones

Las principales consultas a realizar en la base de datos son:

- ¿Cuánta utilidad ha generado un local en un determinado día?

- ¿Qué repartidor ha entregado más deliverys en un mes?
- ¿Cuál es el producto más vendido en un mes?
- ¿Qué cliente ha pedido más deliverys en un mes?
- ¿Cuántas compras han sido realizadas con cada método de pago?
- ¿Cuáles son los productos menos vendidos?
- ¿Cuánta es la proporción de ingresos local/delivery?
- ¿Cuántas compras cuyo monto excede los 20 soles se realizaron en cada mes del año 2021?

Asimismo, se deben considerar las siguientes actualizaciones:

- Actualizaciones automáticas en el stock de un producto en un determinado local al registrar alguna venta del mismo.
- Actualizaciones manuales que registren las entregas de productos de los proveedores.

1.5.5. Tamaño estimado de la base de datos

Mediante los datos brindados por la bodega Jessica, se puede obtener la proyección de bytes del primer mes de creación de la base de datos:

Nombre de la tabla	Longitud de atributos en Bytes	Longitud del registro en Bytes
Persona	8+50	58
Trabajador	8+4+9	21
ClienteDelivery	8+9	17
ClienteLocal	8	8
Turno	8+1+8+8	25
Repartidor	8+50	58
TrabajadorLocal	8+50+20	78
Local	50+4+50+9	113
Caja	50+2	52
Producto	30+20+8	58
Empresa	11+30	41
Compra	4+8+10+8	30
Delivery	4+8+8+50+8+8	86
CompraLocal	4	4
ItemCompra	30+20+4+4+8	66
Stock	30+20+50+4	104
EntregaProductos	30+20+11+4+4+50	119
Boleta	4+8	12
Factura	4+11	15
EncargadoCaja	8+2+50	60

Figura 1: Estimación de la Base de Datos

Nombre de la tabla	Tamaño en Bytes	Datos Estimados	Tamaño Total en Bytes
Trabajador	21	12	420
Turno	25	12	300
Repartidor	58	4	232
TrabajadorLocal	58	8	928
Local	113	2	226
Caja	52	6	312
Producto	58	250	14 500
Stock	104	500	52 000
Empresa	41	20	820
EncargadoCaja	60	4	240
Total	590	818	69 978

Figura 2: Tamaño de las tablas fijas

Nombre de la tabla	Tamaño en Bytes	Crecimiento de datos por día	Crecimiento de datos anual	Tamaño anual de Bytes
Persona	58	100	365 000	2 117 000
ClienteDelivery	17	20	7 200	122 400
ClienteLocal	8	80	28 800	230 400
Compra	30	100	36 000	1 080 000
Delivery	86	20	7 200	619 200
CompraLocal	4	80	28 800	115 200
ItemCompra	66	500	180 000	11 880 000
EntregaProductos	119	300	109500	13 030 500
Boleta	12	80	28 800	345 600
Factura	15	10	3 600	54 000
Total	415	1290	793300	29 594 300

Figura 3: Tamaño de las tablas cambiantes

(*) Se está considerando que el crecimiento de los registros de nuevos clientes es lineal.

1.6. Objetivos del proyecto

Nuestro proyecto cuenta con 3 objetivos principales:

1. Modelar una base de datos robusta para un negocio real, teniendo en cuenta los requerimientos presentes y futuros que pudiera tener, pues está en proceso de expansión.
2. Mejorar el manejo de información de las compras y ventas del negocio, lo cual permita un cálculo eficiente de los ingresos y gastos de cada local para tomar mejores decisiones de negocios.
3. Implementar un sistema eficiente que permita gestionar entregas a domicilio para optimizar el tiempo de los repartidores.

1.7. Referencias del proyecto

El proyecto se basó en la información brindada por nuestro contacto con la empresa Comercial Jessicacón RUC 10088993484.

1.8. Eventualidades

1.8.1. Problemas que pudieran encontrarse en el proyecto.

- Como uno de los problemas encontrados está que se roben los productos de la tienda y que los números de stock no cuadren en la base datos correctamente, en relación a los productos vendidos con los adquiridos.

1.8.2. Límites y alcances del proyecto

- **Límites:**

No poder registrar de forma efectiva las pérdidas externas del negocio, como los robos de los productos.

- **Alcances:**

Este proyecto diseña un negocio familiar y optimiza sus datos. También permite registrar las ventas por delivery o local de manera efectiva. Además se tienen los datos adicionales de cada producto y su stock en cada local. Este modelo se podría generalizar para los demás negocios, con una modificación del mismo, porque se tiene todas las características.

2. Modelo Entidad-Relación

2.1. Reglas semánticas

- Una persona tiene nombre y es identificada por un DNI.
- Un cliente de un local puede realizar una o más compras generando una boleta.
- Un trabajador es una persona y tiene sueldo y un número de celular.
- Cada trabajador tiene al menos un turno de trabajo, este es caracterizado por el día y es único para cada trabajador, también consta de una hora de entrada y de salida.
- Los repartidores y trabajadores locales son trabajadores.
- Un trabajador local trabaja en exactamente un local; además, un trabajador local puede ser el encargado de una sola caja registradora.
- Cada repartidor trabaja en exactamente un local.
- Un local tiene aforo, nombre, teléfono y se identifica por su dirección.
- Un local puede tener varios repartidores.
- Un local tiene al menos un trabajador de local.
- Una caja registradora se identifica por su número de caja.
- Una caja tiene un único trabajador de local encargado de ella.
- Cada caja es única en el contexto de cada local (dos locales pueden tener una caja número 1 sin ser la misma caja).
- Un producto tiene un precio unitario, un tipo de medida y un nombre que lo identifica.

- Cada local tiene un stock determinado de sus productos (se asume que un local tiene más de un producto).
- Una compra es efectuada en una fecha y hora determinada, esta es identificada por un número correlativo. Además, se requiere conocer el método de pago para dicha compra.
- Un cliente de delivery es una persona de la que se requiere su número de celular. Esta persona puede solicitar deliverys a través de WhatsApp.
- Un delivery es una compra y este tiene una dirección de envío, un costo de envío, una fecha y una hora de entrega del delivery.
- Un delivery es pedido por exactamente un cliente de delivery.
- Un delivery es entregado por exactamente un repartidor.
- Un repartidor entrega más de un delivery.
- Una caja registradora registra al menos una compra en un local y cada compra en un local es registrada por exactamente una caja registradora.
- Por cada compra se requiere saber la cantidad de productos comprados y el subtotal.
- Una empresa tiene un RUC y una razón social.
- Una empresa puede realizar compras mediante una factura.
- Una entrega existe únicamente en el contexto de una empresa y tiene una fecha de entrega y una cantidad determinada de productos a entregar.
- Una entrega actualiza el stock de exactamente un producto.
- El stock de un producto es actualizado por más de una entrega.

2.2. Modelo Entidad-Relación

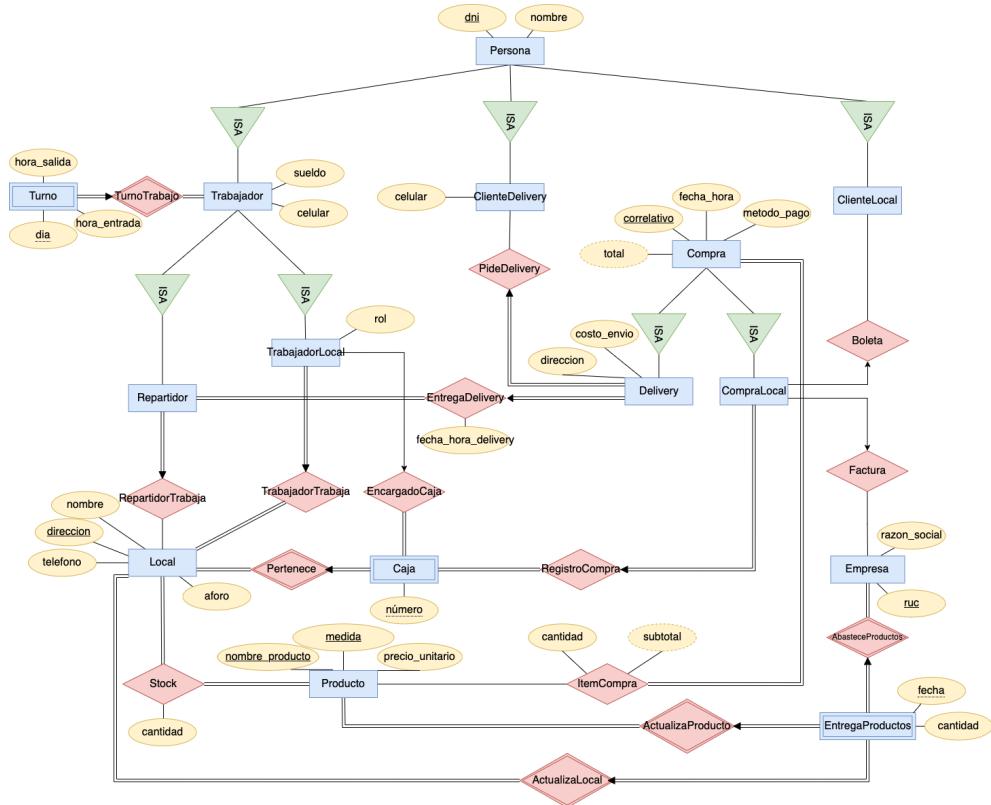


Figura 4: Modelo entidad-relación

2.3. Especificaciones y consideraciones sobre el modelo

2.3.1. Entidad Persona:

■ Especificaciones:

Almacena la información más importante de cada individuo: DNI (como llave primaria) y nombre.

■ Consideraciones:

La llave primaria seleccionada es el DNI; si bien es cierto que nombre es una posible llave candidata, el DNI permite identificar sin dificultades al individuo. Esta entidad es una superclase, la cual hereda a las subclases Trabajador, ClienteDelivery y ClienteLocal. Se permite solapamiento y cobertura.

2.3.2. Entidad Trabajador:

- **Especificaciones:**

Almacena la información básica del trabajador. Esta entidad posee una llave foránea de la entidad persona (DNI). De igual forma, tiene una cantidad de sueldo y un número de celular.

- **Consideraciones:**

La llave principal es el DNI de cada Trabajador, al igual que la entidad anterior, es más fácil de identificar lo hace apto como llave foránea para la tabla Persona. Esta es una subclase que hereda los atributos de Persona. Asimismo, esta es una superclase que hereda a las subclases Repartidor y TrabajadorLocal. Se permite solapamiento y cobertura.

2.3.3. Entidad ClienteDelivery:

- **Especificaciones:**

Almacena la información básica del cliente de delivery. Esta entidad posee una llave foránea de la entidad persona (DNI); también, posee un número de celular.

- **Consideraciones:**

La llave principal es el DNI de cada cliente, ya que es más sencillo de identificar al individuo que ordenó el delivery y lo hace apto como llave foránea para la tabla Persona. Esta es una subclase que hereda los atributos de Persona.

2.3.4. Entidad ClienteLocal

- **Especificaciones:**

Almacena la información básica del cliente en el local. Esta entidad posee una llave foránea de la entidad persona (DNI).

- **Consideraciones:**

La llave principal es el DNI de cada cliente, ya que es más sencillo de identificar al individuo que ordenó un pedido en el local y lo hace apto como llave foránea para la tabla Persona. Esta es una subclase que hereda los atributos de Persona.

2.3.5. Entidad Turno:

- **Especificaciones:**

Guarda el DNI del trabajador, el día, la hora de entrada y de salida. La llave compuesta por el día y el DNI del trabajador siendo este último una llave foránea.

- **Consideraciones:**

El atributo día es una llave principal que nos ayudará a identificar cuál

turno le corresponde al trabajador. Además, la llave foránea DNI permite identificarlo fácilmente. Esta entidad es débil, ya que cada trabajador puede poseer uno o varios turnos.

2.3.6. Entidad Repartidor:

- **Especificaciones:**

Almacena el DNI del repartidor y la dirección del local.

- **Consideraciones:**

La llave principal es el DNI del repartidor, debido a que no es complicado identificar a la persona que reparte los pedidos y lo hace apto como llave foránea para la tabla Persona. Esta es una subclase que hereda los atributos de Trabajador.

2.3.7. Entidad TrabajadorLocal:

- **Especificaciones:**

Almacena el DNI del trabajador del local y su dirección. Asimismo, esta entidad posee un rol.

- **Consideraciones:**

La llave principal es el DNI del trabajador del local, debido a que no es complicado identificar a la persona que trabaja en un local y lo hace apto como llave foránea para la tabla Persona. Esta es una subclase que hereda los atributos de Trabajador.

2.3.8. Entidad Local:

- **Especificaciones:**

Almacena la dirección del lugar, su aforo, el nombre y el número telefónico.

- **Consideraciones:**

La llave primaria seleccionada es dirección, ya que la manera de ubicar un establecimiento es mediante el nombre de la calle, urbanización, avenida, jirones, etc.

2.3.9. Entidad Caja:

- **Especificaciones:**

Guarda la dirección del local y un número, que juntas formarán la llave primaria.

- **Consideraciones:**

La llave primaria compuesta permite identificar a la caja mediante su número y la dirección, al ser esta última una llave foránea para la tabla Local. Esta entidad es débil, ya que cada trabajador puede estar en una o varias cajas.

2.3.10. Entidad Producto:

- **Especificaciones:**

Almacena el nombre del producto, su medida y su precio unitario; el nombre y medida serán componentes de la llave primaria compuesta de esta entidad.

- **Consideraciones:**

La llave primaria compuesta permite identificar al producto mediante su nombre y su medida, ya que es muy complicado distinguir un producto por medio del precio unitario.

2.3.11. Entidad Empresa:

- **Especificaciones:**

Guarda el RUC de la empresa y su razón social. Su llave primaria es el RUC.

- **Consideraciones:**

El RUC es considerado llave primaria, debido a que es necesario para reconocer el nombre de la empresa.

2.3.12. Entidad Compra:

- **Especificaciones:**

Almacena el número correlativo, la fecha y hora, el método de pago y el total a pagar; siendo el primero la llave primaria de la relación.

- **Consideraciones:**

El correlativo es óptimo para ser llave primaria, ya que mediante números es más sencillo identificar el orden de la compra y sus especificaciones. Además, esta es una superclase que hereda a las subclases Delivery y CompraLocal.

2.3.13. Entidad Delivery:

- **Especificaciones:**

Guarda el número correlativo de la compra, el DNI del repartidor, el DNI del cliente de delivery, la dirección, el costo de envío y la fecha y hora del delivery. La llave primaria vendría a ser el correlativo.

- **Consideraciones:**

La llave primaria es el correlativo de la compra, esto debido a que el repartidor podrá conocer el número de compra y si desea especificaciones lo consulta con la bodega; por ello es óptimo. Esta es una subclase que hereda los atributos de Compra.

2.3.14. Entidad CompraLocal:

- **Especificaciones:**

Almacena el número correlativo de la compra. Igualmente, esta es la llave primaria foránea de la entidad.

- **Consideraciones:**

Debido a que es una entidad sin atributos propios, su única llave primaria debe ser foránea y en este caso de la entidad compra. Esta es una subclase que hereda los atributos de Compra.

2.3.15. Entidad EntregaProductos:

- **Especificaciones:**

Almacena el nombre del producto, su medida, el RUC de la empresa, la dirección del local, fecha y cantidad. La llave primaria esta compuesta por nombre, medida, RUC, dirección y fecha; los cuatro primeros atributos son llaves foráneas.

- **Consideraciones:**

La llave primaria compuesta permite identificar la entrega de los productos mediante el nombre del producto, su medida, el RUC de la empresa, la dirección y la fecha de compra. Los cuatro primeros atributos son llaves foráneas para las tablas Producto, Empresa y Local. Esta entidad es débil, ya que cada entrega puede darse en una o varios locales.

3. Modelo Relacional

3.1. Modelo Relacional

- **Persona(dni, nombre)**
- **Trabajador(Persona.dni, sueldo, celular)**
- **ClienteDelivery(Persona.dni, celular)**
- **ClienteLocal(Persona.dni)**
- **Turno(Trabajador.dni, día, hora_entrada, hora_salida)**
- **Repartidor(Trabajador.dni, Local.direccion)**
- **TrabajadorLocal(Trabajador.dni, Local.direccion, rol)**
- **Local(direccion, aforo, nombre, telefono)**
- **Caja(Local.direccion, numero)**
- **Producto(nombre_producto, medida, precio_unitario)**
- **Empresa(ruc, razon_social)**

- **Compra**(correlativo, fecha_hora, metodo_pago, total)
- **Delivery**(Compra.correlativo, Repartidor.dni, ClienteDelivery.dni, direccion, costo_envio, fecha_hora_delivery)
- **CompraLocal**(Compra.correlativo, Caja.numero, Caja.direccion)
- **ItemCompra**(Producto.nombre_producto, Producto.medida, Compra.correlativo, cantidad, subtotal)
- **Stock**(Producto.nombre_producto, Producto.medida Local.direccion, cantidad)
- **EntregaProductos**(Producto.nombre_producto, Producto.medida, Empresa.ruc, Local.direccion, fecha, cantidad)
- **Boleta**(CompraLocal.correlativo, ClienteLocal.dni)
- **Factura**(CompraLocal.correlativo, Empresa.ruc)
- **EncargadoCaja**(TrabajadorLocal.dni, Caja.numero, Caja.direccion)

3.2. Especificaciones de transformación

3.2.1. Entidades

Para transformar las entidades se han tenido en cuenta las siguientes especificaciones:

- **Local:** Su llave primaria es dirección y tiene como atributos el nombre del local, su aforo y un teléfono asociado.
- **Producto:** Su llave primaria es una llave compuesta por el nombre del producto y la unidad de medida utilizada para medir la cantidad del producto. Además, tiene el precio unitario, que se refiere al precio de una unidad de la medida designada para el producto con dicho nombre.
- **Empresa:** Tiene una razón social y su llave primaria es su RUC.

3.2.2. Entidades débiles

- **Turno:** Es entidad débil de Trabajador, ya que solo existe en el contexto dentro de un Trabajador. Su llave primaria la conforman día y DNI del Trabajador. Tiene como atributos hora_de_entrada y hora_de_salida.
- **Entrega Productos:** Es entidad débil tanto de Producto, Empresa como de Local, debido a que solo existe cuando una Empresa hace una entrega de productos a cierto local. Su llave primaria la conforman nombre_de_producto, medida, local, RUC y fecha. Tiene como atributo cantidad.

- **Caja:** Es entidad débil de Local, ya que una Caja solo existe dentro del contexto de un solo Local. Su llave primaria está conformada por dirección de Local y número de Caja.

3.2.3. Entidades superclase/subclase

- **Superclase (Persona):** Esta superclase cuenta con los atributos DNI y nombre. La llave primaria es DNI y hay tres entidades que heredan de esta entidad. Hay solapamiento, por lo que la entidad persona se transforma en una tabla. Sí hay cobertura, pues las personas en el modelo deben de ser trabajadores, clientes de delivery o clientes de local.

- **Subclases**

- **Trabajador:** Un trabajador puede ser también cliente en algún otro local, por lo que hay solapamiento. Se crea una tabla para esta entidad con DNI de persona como llave y llave foránea.
- **ClienteDelivery:** Por el solapamiento, se crea una tabla para esta entidad con DNI como llave y llave foránea, además, cuenta con el atributo celular.
- **ClienteLocal:** Se crea una tabla para esta entidad, pues hay solapamiento. Esta tiene como único atributo al DNI de la persona, y lo hereda como llave y llave foránea.
- **Superclase (Trabajador):** La llave primaria es DNI, heredada de Persona. No hay solapamiento, pues un repartidor no puede trabajar en un local. En el contexto del problema sí hay cobertura.

- **Subclases**

- **TrabajadorLocal:** No hay solapamiento; sin embargo, se crea una tabla con DNI de trabajador como llave y llave foránea por si en algún momento se requiriese tener solapamiento, es decir, para que la base de datos sea fácilmente escalable.
- **Repartidor:** Al igual que con TrabajadorLocal, se crea una tabla con DNI de trabajador como llave y llave foránea para que la base de datos sea escalable.
- **Superclase (Compra):** No hay solapamiento. Sí hay cobertura. La llave primaria es correlativo.

- **Subclases**

- **Delivery:** No hay solapamiento; sin embargo, se crea una tabla con correlativo como llave y llave foránea por si en algún momento se requiriese tener solapamiento, es decir, para que la base de datos sea fácilmente escalable.

- **CompraLocal:** Al igual que con Delivery, se crea una tabla con correlativo como llave y llave foránea para que la base de datos sea escalable.

3.2.4. Relaciones binarias

- **ItemCompra:** Esta relación se lleva a cabo entre Compra y Producto. Compra tiene una multiplicidad de 1 a n, ya que se debe comprar mínimo un producto, y se pueden comprar n productos. Producto tiene una multiplicidad de 0 a n, ya que puede ser que no sea comprado en una compra, o sea comprado en varias compras.
- **Stock:** Esta relación se lleva a cabo entre Producto y Local. Producto tiene una multiplicidad de 1 a n porque para entrar en la base de datos, debe existir en mínimo en 1 local y puede existir en varios. Local también una multiplicidad de 1 a n porque un local tiene varios productos.
- **Factura:** Esta relación se lleva a cabo entre Empresa y CompraLocal. Empresa tiene una multiplicidad de 0 a n productos, ya que no necesariamente una Empresa genera una factura para hacer una compra en un local y puede generar varias facturas. CompraLocal tiene una multiplicidad de 0 a 1, ya que una compra puede o no ser de tipo factura.
- **Boleta:** Esta relación se lleva a cabo entre ClienteLocal y CompraLocal. Persona tiene una multiplicidad de 0 a n productos, ya que no necesariamente una persona genera una boleta para hacer una compra, pero si hiciera más de una compra podría generar varias boletas. Compra tiene una multiplicidad de 0 a 1, ya que una compra puede o no ser de tipo boleta.
- **EncargadoCaja:** Esta relación se lleva a cabo entre Caja y TrabajadorLocal. Caja tiene una multiplicidad de 1 a n, ya que una caja puede tener varios encargados en diferentes turnos. TrabajadorLocal tiene una multiplicidad de 0 a 1, ya que un trabajador puede ser o no encargado de la caja.

3.2.5. Relaciones ternarias

Para la elaboración del modelo no se consideraron relaciones ternarias.

3.3. Diccionario de datos

Persona				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
dni	VARCHAR(8)	X		DNI de la persona
nombre	VARCHAR(50)			Nombre de la persona

Figura 5: Tabla Persona

Trabajador				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Persona.dni	VARCHAR(8)	X	X	DNI del Trabajador heredado de la entidad Persona
sueldo	DOUBLE PRECISION			Sueldo del trabajador
celular	VARCHAR(9)			Celular del trabajador

Figura 6: Tabla Trabajador

ClienteDelivery				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Persona.dni	VARCHAR(8)	X	X	DNI del cliente delivery heredado de la entidad Persona
celular	VARCHAR(9)			Celular del cliente delivery

Figura 7: Tabla ClienteDelivery

ClienteLocal				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Persona.dni	VARCHAR(8)	X	X	DNI del cliente delivery heredado de la entidad Persona

Figura 8: Tabla ClienteLocal

Turno				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Trabajador.dni	VARCHAR(8)	X	X	DNI del trabajador del turno heredado de la entidad Persona
dia	VARCHAR(1)	X		Día de la semana que trabaja el trabajador (L, M, X, J, V, S, D)
hora_entrada	TIME			Hora de entrada del turno
hora_salida	TIME			Hora de salida del turno

Figura 9: Tabla Turno

Repartidor				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Trabajador.dni	VARCHAR(8)	X	X	DNI del repartidor heredado de la entidad Trabajador
Local.direccion	VARCHAR(50)		X	Dirección del local donde trabaja el repartidor

Figura 10: Tabla Repartidor

TrabajadorLocal				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Trabajador.dni	VARCHAR(8)	X	X	DNI del trabajador local heredado de la entidad Trabajador
Local.direccion	VARCHAR(50)		X	Dirección del local donde trabaja el trabajador local
rol	VARCHAR(20)			Rol que cumple el trabajador local

Figura 11: Tabla TrabajadorLocal

Local				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
direccion	VARCHAR(50)	X		Dirección del local
aforo	SMALLINT			Aforo del local
nombre	VARCHAR(50)			Nombre del local
telefono	VARCHAR(9)			Número de teléfono del local

Figura 12: Tabla Local

Caja				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Local.direccion	VARCHAR(50)	X	X	Dirección del local al que pertenece la caja
numero	SMALLINT	X		Número de caja

Figura 13: Tabla Caja

Producto				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
nombre_producto	VARCHAR(30)	X		Nombre que identifica al producto
medida	VARCHAR(20)	X		Medida la presentación del producto
precio_unitario	DECIMAL			Precio unitario del producto

Figura 14: Tabla Producto

Empresa				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
ruc	VARCHAR(11)	X		RUC de la empresa
razon_social	VARCHAR(30)			Razón social de la empresa

Figura 15: Tabla Empresa

Compra				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
correlativo	INTEGER	X		Número que identifica a la compra
fecha_hora	DATETIME			Fecha y hora de la realización de la compra
metodo_pago	VARCHAR(10)			Método de pago de la compra
total	DECIMAL			Precio total de la compra que es un atributo derivado

Figura 16: Tabla Compra

Delivery				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Compra.correlativo	INTEGER	X	X	Número que identifica a la compra
Repartidor.dni	VARCHAR(8)		X	DNI que identifica al repartidor encargado del delivery
ClienteDelivery.dni	VARCHAR(8)		X	DNI que identifica al cliente que pidió el delivery
direccion	VARCHAR(50)			Dirección de la entrega del delivery
costo_envio	SMALLINT			Precio del costo de envío
fecha_hora_delivery	DATETIME			Fecha y hora de llegada del delivery

Figura 17: Tabla Delivery

CompraLocal				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Compra.correlativo	INTEGER	X	X	Número que identifica a la compra en el local
Caja.numero	SMALLINT		X	Número que identifica a la caja en el local
Caja.direccion	VARCHAR(50)		X	Dirección que identifica al local donde se encuentra la caja

Figura 18: Tabla CompraLocal

ItemCompra				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Producto.nombre_producto	VARCHAR(30)	X	X	Nombre del producto que se compra en el local
Producto.medida	VARCHAR(20)	X	X	Medida de la presentación del producto que se compra en el local
Compra.correlativo	INTEGER	X	X	Número que identifica a la compra
cantidad	INTEGER			Cantidad del producto comprado en la compra
subtotal	DECIMAL			Precio subtotal de la compra que es un atributo derivado

Figura 19: Tabla ItemCompra

Stock				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Producto.nombre_producto	VARCHAR(30)	X	X	Nombre del producto
Producto.medida	VARCHAR(20)	X	X	Medida de la presentación del producto
Local.direccion	VARCHAR(50)	X	X	Dirección del local donde se encuentra el producto
cantidad	INTEGER			Cantidad del producto en el local

Figura 20: Tabla Stock

EntregaProductos				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
Producto.nombre_producto	VARCHAR(30)	X	X	Nombre del producto que se va a realizar la entrega
Producto.medida	VARCHAR(20)	X	X	Medida de la presentación del producto que se va a realizar la entrega
Local.direccion	VARCHAR(50)	X	X	Local a donde la entrega se va a realizar
Empresa.ruc	VARCHAR(11)	X	X	RUC de la empresa que va a realizar la entrega
fecha	DATE	X		Fecha en la que se va a realizar la entrega del producto
cantidad	INTEGER			Cantidad de productos que se van a entregar

Figura 21: Tabla EntregaProductos

Boleta				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
CompraLocal.correlativo	INTEGER	X	X	Número con el que se identifica a la compra
ClienteLocal.dni	VARCHAR(8)		X	DNI del cliente local

Figura 22: Tabla Boleta

Factura				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
CompraLocal.correlativo	INTEGER	X	X	Número con el que se identifica a la compra
Empresa.ruc	VARCHAR(11)		X	RUC de la empresa

Figura 23: Tabla Factura

EncargadoCaja				
Nombre Campo	Tipo de Dato	PK	FK	Descripción
TrabajadorLocal.dni	VARCHAR(8)	X	X	DNI del trabajador local
Caja.numero	SMALLINT		X	Número con el que se identifica a la caja
Caja.direccion	VARCHAR(50)		X	Dirección del local al que pertenece la caja

Figura 24: Tabla EncargadoCaja

4. Implementación de la base de datos

4.1. Creación de tablas en PostgreSQL

```
1 -- Creacion de tablas
2
3 CREATE TABLE IF NOT EXISTS persona (
4     dni VARCHAR(8),
5     nombre VARCHAR(50)
6 );
7
8 CREATE TABLE IF NOT EXISTS trabajador(
9     dni VARCHAR(8),
10    sueldo DOUBLE PRECISION,
11    celular VARCHAR(9)
12 );
13
14 CREATE TABLE IF NOT EXISTS clientedelivery(
15     dni VARCHAR(8),
16     celular VARCHAR(9)
17 );
18
19 CREATE TABLE IF NOT EXISTS clientelocal(
20     dni VARCHAR(8)
21 );
22
23 CREATE TABLE IF NOT EXISTS turno(
24     dni VARCHAR(8),
25     dia VARCHAR(1),
26     hora_entrada TIME,
27     hora_salida TIME
28 );
29
30 CREATE TABLE IF NOT EXISTS repartidor(
31     dni VARCHAR(8),
32     direccion VARCHAR(50)
33 );
34
35 CREATE TABLE IF NOT EXISTS trabajadorlocal(
36     dni VARCHAR(8),
37     direccion VARCHAR(50),
38     rol VARCHAR(20)
39 );
40
41 CREATE TABLE IF NOT EXISTS local(
42     direccion VARCHAR(50),
43     aforo SMALLINT,
44     nombre VARCHAR(50),
45     telefono VARCHAR(9)
46 );
47
48 CREATE TABLE IF NOT EXISTS caja(
49     direccion VARCHAR(50),
50     numero SMALLINT
51 );
52
53
```

```

54 CREATE TABLE IF NOT EXISTS producto (
55     nombre_producto VARCHAR(30),
56     medida VARCHAR(20),
57     precio_unitario DECIMAL
58 );
59
60 CREATE TABLE IF NOT EXISTS empresa (
61     ruc VARCHAR(11),
62     razon_social VARCHAR(30)
63 );
64
65 CREATE TABLE IF NOT EXISTS compra (
66     correlativo INTEGER,
67     fecha_hora TIMESTAMP,
68     metodo_pago VARCHAR(10),
69     total DECIMAL
70 );
71
72 CREATE TABLE IF NOT EXISTS delivery (
73     correlativo INTEGER,
74     Rdni VARCHAR(8),
75     Cdni VARCHAR(8),
76     direccion VARCHAR(50),
77     costo_envio SMALLINT,
78     fecha_hora_delivery TIMESTAMP
79 );
80
81 CREATE TABLE IF NOT EXISTS compralocal (
82     correlativo INTEGER,
83     numero SMALLINT,
84     direccion VARCHAR(50)
85 );
86
87 CREATE TABLE IF NOT EXISTS itemcompra (
88     nombre_producto VARCHAR(30),
89     medida VARCHAR(20),
90     correlativo INTEGER,
91     cantidad INTEGER,
92     subtotal DECIMAL
93 );
94
95 CREATE TABLE IF NOT EXISTS stock (
96     nombre_producto VARCHAR(30),
97     medida VARCHAR(20),
98     direccion VARCHAR(50),
99     cantidad INTEGER
100 );
101
102 CREATE TABLE IF NOT EXISTS entregaproductos (
103     nombre_producto VARCHAR(30),
104     medida VARCHAR(20),
105     direccion VARCHAR(50),
106     ruc VARCHAR(11),
107     fecha DATE,
108     cantidad INTEGER
109 );
110

```

```

111 CREATE TABLE IF NOT EXISTS boleta (
112     correlativo INTEGER ,
113     dni VARCHAR(8)
114 );
115
116 CREATE TABLE IF NOT EXISTS factura (
117     correlativo INTEGER ,
118     ruc VARCHAR(11)
119 );
120
121 CREATE TABLE IF NOT EXISTS encargadocaja(
122     dni VARCHAR(8),
123     numero SMALLINT,
124     direccion VARCHAR(50)
125 );
126
127
128 -- Key constraints
129
130 -- Persona
131 ALTER TABLE persona ADD CONSTRAINT pk_persona_dni PRIMARY KEY (dni)
132 ;
133
134 -- Trabajador
135 ALTER TABLE Trabajador ADD CONSTRAINT pk_trabajador_dni PRIMARY KEY
(dni);
136 ALTER TABLE Trabajador ADD CONSTRAINT fk_trabajador_persona_dni
FOREIGN KEY (dni) REFERENCES persona(dni);
137
138
139 -- ClienteDelivery
140 ALTER TABLE ClienteDelivery ADD CONSTRAINT pk_cliente_delivery_dni
PRIMARY KEY (dni);
141 ALTER TABLE ClienteDelivery ADD CONSTRAINT
fk_cliente_delivery_persona_dni FOREIGN KEY (dni) REFERENCES
persona(dni);
142
143
144 -- ClienteLocal
145 ALTER TABLE ClienteLocal ADD CONSTRAINT pk_cliente_local_dni
PRIMARY KEY (dni);
146 ALTER TABLE ClienteLocal ADD CONSTRAINT
fk_cliente_local_persona_dni FOREIGN KEY (dni) REFERENCES
persona(dni);
147
148
149 -- Turno
150 ALTER TABLE Turno ADD CONSTRAINT pk_turno_dni_dia PRIMARY KEY (dni ,
dia);
151 ALTER TABLE Turno ADD CONSTRAINT fk_turno_trabajador_dni FOREIGN
KEY (dni) REFERENCES Trabajador(dni);
152
153
154 -- LOCAL
155 ALTER TABLE Local ADD CONSTRAINT pk_local_direccion PRIMARY KEY (
direccion);

```

```

156
157
158 -- Repartidor
159 ALTER TABLE Repartidor ADD CONSTRAINT pk_repartidor_dni PRIMARY KEY
   (dni);
160 ALTER TABLE Repartidor ADD CONSTRAINT fk_repartidor_trabajador_dni
   FOREIGN KEY (dni) REFERENCES Trabajador(dni);
161 ALTER TABLE Repartidor ADD CONSTRAINT fk_repartidor_local_direccion
   FOREIGN KEY (direccion) REFERENCES local(direccion);
162
163
164 -- TrabajadorLocal
165 ALTER TABLE TrabajadorLocal ADD CONSTRAINT pk_trabajador_local_dni
   PRIMARY KEY (dni);
166 ALTER TABLE TrabajadorLocal ADD CONSTRAINT
   fk_trabajador_local_trabajador_dni FOREIGN KEY (dni) REFERENCES
   Trabajador(dni);
167 ALTER TABLE TrabajadorLocal ADD CONSTRAINT
   fk_trabajador_local_local_direccion FOREIGN KEY (direccion)
   REFERENCES local(direccion);
168
169
170
171 -- Caja
172 ALTER TABLE Caja ADD CONSTRAINT pk_caja_direccion_numero PRIMARY
   KEY (direccion, numero);
173 ALTER TABLE Caja ADD CONSTRAINT fk_caja_local_direccion FOREIGN KEY
   (direccion) REFERENCES Local(direccion);
174
175
176 -- Producto
177 ALTER TABLE Producto ADD CONSTRAINT pk_producto_nombre_medida
   PRIMARY KEY (nombre_producto, medida);
178
179
180 -- Empresa
181 ALTER TABLE Empresa ADD CONSTRAINT pk_empresa_ruc PRIMARY KEY (ruc)
   ;
182
183
184 -- Compra
185 ALTER TABLE Compra ADD CONSTRAINT pk_compra_correlativo PRIMARY KEY
   (correlativo);
186
187
188 -- Delivery
189 ALTER TABLE Delivery ADD CONSTRAINT pk_delivery_correlativo PRIMARY
   KEY (correlativo);
190 ALTER TABLE Delivery ADD CONSTRAINT fk_delivery_compra_correlativo
   FOREIGN KEY (correlativo) REFERENCES Compra(correlativo);
191 ALTER TABLE Delivery ADD CONSTRAINT fk_delivery_repartidor_dni
   FOREIGN KEY (Rdni) REFERENCES Repartidor(dni);
192 ALTER TABLE Delivery ADD CONSTRAINT fk_delivery_cliente_dni FOREIGN
   KEY (Cdni) REFERENCES clientedelivery(dni);
193
194
195 -- Compra Local

```

```

196 ALTER TABLE CompraLocal ADD CONSTRAINT pk_compra_local_correlativo
    PRIMARY KEY (correlativo) ;
197 ALTER TABLE CompraLocal ADD CONSTRAINT
    fk_compra_local_compra_correlativo FOREIGN KEY (correlativo)
    REFERENCES Compra(correlativo);
198 ALTER TABLE CompraLocal ADD CONSTRAINT fk_compra_local_caja_numero
    FOREIGN KEY (numero , direccion) REFERENCES Caja(numero ,
    direccion);
199
200
201 -- Item Compra
202 ALTER TABLE ItemCompra ADD CONSTRAINT
    pk_item_compra_nombre_correlativo_medida PRIMARY KEY (
        nombre_producto , correlativo , medida);
203 ALTER TABLE ItemCompra ADD CONSTRAINT
    fk_item_compra_producto_nombre_medida FOREIGN KEY (
        nombre_producto , medida) REFERENCES Producto(nombre_producto ,
        medida);
204 ALTER TABLE ItemCompra ADD CONSTRAINT
    fk_item_compra_compra_correlativo FOREIGN KEY (correlativo)
    REFERENCES Compra(correlativo);
205
206
207 -- Stock
208 ALTER TABLE Stock ADD CONSTRAINT pk_stock_direccion PRIMARY KEY (
    direccion , medida , nombre_producto);
209 ALTER TABLE Stock ADD CONSTRAINT fk_stock_producto_nombre_medida
    FOREIGN KEY (nombre_producto , medida) REFERENCES Producto(
        nombre_producto , medida);
210 ALTER TABLE Stock ADD CONSTRAINT fk_stock_local_direccion FOREIGN
    KEY (direccion) REFERENCES Local(direccion);
211
212
213 -- Entrega Productos
214 ALTER TABLE EntregaProductos ADD CONSTRAINT
    pk_entrega_productos_medida PRIMARY KEY (medida , nombre_producto ,
    direccion , fecha , ruc);
215 ALTER TABLE EntregaProductos ADD CONSTRAINT
    fk_entrega_productos_producto_nombre_medida FOREIGN KEY (
        nombre_producto , medida) REFERENCES Producto (nombre_producto ,
        medida);
216 ALTER TABLE EntregaProductos ADD CONSTRAINT
    fk_entrega_productos_local_direccion FOREIGN KEY (direccion)
    REFERENCES Local(direccion);
217 ALTER TABLE EntregaProductos ADD CONSTRAINT
    fk_entrega_productos_empresa FOREIGN KEY (ruc) REFERENCES
    Empresa(ruc);
218
219
220 -- Boleta
221 ALTER TABLE Boleta ADD CONSTRAINT pk_boleta_correlativo PRIMARY KEY
    (correlativo);
222 ALTER TABLE Boleta ADD CONSTRAINT
    fk_boleta_compra_local_correlativo FOREIGN KEY (correlativo)
    REFERENCES CompraLocal(correlativo);
223 ALTER TABLE Boleta ADD CONSTRAINT fk_boleta_cliente_local_dni
    FOREIGN KEY (dni) REFERENCES clientelocal(dni);

```

```

224
225
226 -- Factura
227 ALTER TABLE Factura ADD CONSTRAINT pk_factura_correlativo PRIMARY
   KEY (correlativo);
228 ALTER TABLE Factura ADD CONSTRAINT
   fk_factura_compra_local_correlativo FOREIGN KEY (correlativo)
   REFERENCES CompraLocal(correlativo);
229 ALTER TABLE Factura ADD CONSTRAINT fk_factura_empresa_ruc FOREIGN
   KEY (ruc) REFERENCES Empresa(ruc);
230
231
232 -- Encargado Caja
233 ALTER TABLE encargadocaja ADD CONSTRAINT pk_encargado_caja_dni
   PRIMARY KEY (dni);
234 ALTER TABLE encargadocaja ADD CONSTRAINT
   fk_encargado_caja_trabajador_local_dni FOREIGN KEY (dni)
   REFERENCES trabajadorlocal(dni);
235 ALTER TABLE encargadocaja ADD CONSTRAINT
   fk_encargado_caja_caja_numero_direccion FOREIGN KEY (numero,
   direccion) REFERENCES Caja(numero, direccion);
236
237
238 -- Not null constraints
239
240 ALTER TABLE persona ALTER COLUMN nombre SET NOT NULL;
241
242 ALTER TABLE trabajador ALTER COLUMN sueldo SET NOT NULL ;
243 ALTER TABLE trabajador ALTER COLUMN celular SET NOT NULL ;
244
245 ALTER TABLE clientedelivery ALTER COLUMN celular SET NOT NULL ;
246
247 ALTER TABLE turno ALTER COLUMN hora_entrada SET NOT NULL ;
248 ALTER TABLE turno ALTER COLUMN hora_salida SET NOT NULL ;
249
250 ALTER TABLE trabajadorlocal ALTER COLUMN rol SET NOT NULL ;
251
252 ALTER TABLE local ALTER COLUMN direccion SET NOT NULL ;
253
254 ALTER TABLE producto ALTER COLUMN precio_unitario SET NOT NULL ;
255
256 ALTER TABLE empresa ALTER COLUMN razon_social SET NOT NULL ;
257
258 ALTER TABLE compra ALTER COLUMN fecha_hora SET NOT NULL ;
259 ALTER TABLE compra ALTER COLUMN metodo_pago SET NOT NULL ;
260 ALTER TABLE compra ALTER COLUMN total SET NOT NULL ;
261
262 ALTER TABLE delivery ALTER COLUMN direccion SET NOT NULL ;
263 ALTER TABLE delivery ALTER COLUMN costo_envio SET NOT NULL ;
264 ALTER TABLE delivery ALTER COLUMN fecha_hora_delivery SET NOT NULL
   ;
265
266 ALTER TABLE itemcompra ALTER COLUMN cantidad SET NOT NULL ;
267 ALTER TABLE itemcompra ALTER COLUMN subtotal SET NOT NULL ;
268
269 ALTER TABLE stock ALTER COLUMN cantidad SET NOT NULL ;
270

```

```

271 ALTER TABLE entregaproductos ALTER COLUMN cantidad SET NOT NULL ;
272
273 -- Otros constraints
274
275
276 -- Empresa
277 ALTER TABLE empresa ADD CONSTRAINT unique_empresa_razon_social
    UNIQUE(razon_social);
278
279 -- Empresa
280 ALTER TABLE empresa ADD CONSTRAINT unique_empresa_razon_social
    UNIQUE(razon_social);
281
282 -- Producto
283 ALTER TABLE producto ADD CONSTRAINT producto_precio_unitario CHECK
    (precio_unitario > 0);
284
285 -- Compra
286 ALTER TABLE compra ADD CONSTRAINT compra_metodo_pago CHECK (
    metodo_pago IN ('Yape', 'Plin', 'Visa', 'MasterCard', 'Efectivo
    '));
287
288 -- Stock
289 ALTER TABLE stock ADD CONSTRAINT stock_cantidad CHECK (cantidad >=
    0);
290
291 -- Local
292 ALTER TABLE local ADD CONSTRAINT local_aforo CHECK (aforo > 0);
293
294 -- Turno
295 ALTER TABLE turno ADD CONSTRAINT turno_dia CHECK (dia IN ('L', 'M',
    'X', 'J', 'V', 'S', 'D'));
296 ALTER TABLE turno ADD CONSTRAINT turno_hora_entrada_salida CHECK (
    hora_entrada < hora_salida);
297
298 -- Trabajador
299 ALTER TABLE trabajador ADD CONSTRAINT trabajador_sueldo CHECK (
    sueldo > 0);
300
301 -- Triggers
302 CREATE OR REPLACE FUNCTION actualizar_subtotal()
303 RETURNS TRIGGER AS
304 $$$
305     BEGIN
306         UPDATE itemcompra i SET subtotal = NEW.cantidad * (
307             SELECT precio_unitario
308             FROM producto p
309             WHERE p.nombre_producto = NEW.nombre_producto
310             AND p.medida = NEW.medida
311         )
312             WHERE i.nombre_producto = NEW.nombre_producto
313             AND i.medida = NEW.medida;
314         RETURN NEW;
315     END;
316 $$ LANGUAGE plpgsql;
317
318 CREATE TRIGGER actualizar_subtotal

```

```

319 AFTER INSERT ON itemcompra
320 FOR EACH ROW EXECUTE FUNCTION actualizar_subtotal();
321
322
323 CREATE OR REPLACE FUNCTION actualizar_total()
324 RETURNS TRIGGER AS
325 $$ 
326 BEGIN
327   UPDATE compra SET total = (
328     SELECT SUM(subtotal)
329     FROM itemcompra
330    WHERE NEW.correlativo = itemcompra.correlativo
331   )
332   WHERE correlativo = NEW.correlativo;
333   RETURN NEW;
334 END;
335 $$ LANGUAGE plpgsql;
336
337 CREATE TRIGGER actualizar_total
338 AFTER INSERT ON itemcompra
339 FOR EACH ROW EXECUTE FUNCTION actualizar_total();
340
341 CREATE OR REPLACE FUNCTION sumar_stock()
342 RETURNS TRIGGER AS
343 $$ 
344 BEGIN
345   IF EXISTS (
346     SELECT cantidad
347       FROM stock
348      WHERE nombre_producto = NEW.nombre_producto
349      AND medida = NEW.medida
350      AND direccion = NEW.direccion)
351   THEN
352     UPDATE stock SET cantidad = cantidad + NEW.cantidad
353     WHERE nombre_producto = NEW.nombre_producto
354     AND medida = NEW.medida
355     AND direccion = NEW.direccion;
356   ELSE
357     INSERT INTO stock(nombre_producto, medida, direccion,
358   cantidad) VALUES
359     (NEW.nombre_producto, NEW.medida, NEW.direccion,
360     NEW.cantidad);
361   END IF;
362   RETURN NEW;
363 END;
364 $$ LANGUAGE plpgsql;
365
366 CREATE TRIGGER sumar_stock
367 AFTER INSERT ON entregaproductos
368 FOR EACH ROW EXECUTE FUNCTION sumar_stock();
369
370 CREATE OR REPLACE FUNCTION restar_stock()
371 RETURNS TRIGGER AS
372 $$ 
373 BEGIN
374   IF EXISTS (

```

```

374     SELECT direccion
375     FROM compralocal
376     WHERE correlativo = NEW.correlativo)
377 THEN
378     UPDATE stock SET cantidad = cantidad - NEW.cantidad
379     WHERE nombre_producto = NEW.nombre_producto
380     AND medida = NEW.medida
381     AND direccion = (
382         SELECT direccion
383         FROM compralocal
384         WHERE correlativo = NEW.correlativo
385     );
386 ELSIF EXISTS (
387     SELECT direccion
388     FROM delivery
389     WHERE correlativo = NEW.correlativo)
390 THEN
391     UPDATE stock SET cantidad = cantidad - NEW.cantidad
392     WHERE nombre_producto = NEW.nombre_producto
393     AND medida = NEW.medida
394     AND direccion = (
395         SELECT repartidor.direccion
396         FROM delivery JOIN repartidor
397         ON delivery.rdni = repartidor.dni
398         WHERE NEW.correlativo = delivery.correlativo
399     );
400 ELSE
401     UPDATE stock SET cantidad = cantidad - NEW.cantidad
402     WHERE nombre_producto = NEW.nombre_producto
403     AND medida = NEW.medida
404     AND direccion = 'Av. Talara, Cercado de Lima 15836';
405 END IF;
406 RETURN NEW;
407
408 $$ LANGUAGE plpgsql;
409
410 CREATE TRIGGER restar_stock
411 BEFORE INSERT ON itemcompra
412 FOR EACH ROW EXECUTE FUNCTION restar_stock();
413
414 -- Vistas
415 CREATE ROLE repartidor_rol WITH LOGIN;
416
417 CREATE VIEW vista_repartidor AS
418 SELECT nombre, correlativo, direccion, costo_envio,
419     fecha_hora_delivery, metodo_pago, total
420 FROM delivery NATURAL JOIN compra JOIN persona ON (rdni=dni);
421 GRANT SELECT ON vista_repartidor TO repartidor_rol;

```

4.2. Carga de datos

La carga de datos a los esquemas de 1k, 10k, 100k y 1M se realizó junto a la simulación de datos faltantes, puesto que a pesar de que se está modelando una empresa real, la información no fue suficiente para llenar todas las tablas.

4.3. Simulación de datos faltantes

Para los datos faltantes se usó el Software DBeaver Enterprise Edition, la cual posee la herramienta Mock Data que permite generar datos aleatorios que cumplen restricciones de las tablas usando expresiones regulares. A pesar de ser una herramienta muy útil y amigable, tiene ciertas limitaciones al momento de generar foreign keys, por lo que se complementó el uso de la herramienta DBeaver con scripts de Python. En específico, se emplearon las librerías psycopg2, random y datetime con el fin de mantener la integridad de la base de datos.

5. Optimización y experimentación

En la siguiente sección, se probará el rendimiento de la base de datos con 3 consultas complejas. Para esto, se probarán las consultas en diferentes escenarios, con diferentes cantidades de datos y con o sin índices. Asimismo, se crearán los índices que sean necesarios para garantizar la ejecución óptima de estas consultas. Finalmente, se analizarán y compararán los resultados obtenidos.

5.1. Consultas SQL para el experimento

5.1.1. Descripción del tipo de consultas seleccionadas

5.1.1.1. Consulta 1 ¿Cuántas compras cuyo monto excede el promedio se realizaron en cada mes del año 2021 con tarjeta en el local con más ventas?

Justificación: Se requiere saber la cantidad de compras grandes realizadas por los clientes en el local principal para determinar si vale la pena adquirir el nuevo sistema de IziPay.

5.1.1.2. Consulta 2 ¿Cuáles son los 10 bebidas menos y 10 bebidas más vendidas (si los hay) en los meses de la temporada de verano (primer y últimos trimestres del año) de los productos de la empresa que abasteció mayor cantidad de veces a la cadena de locales entre los años 2020 y 2021?

Justificación: La empresa desea saber qué bebidas son las menos vendidas para así la siguiente temporada evaluar si se desea adquirir tales productos. Para lograr ello, se desea saber un total de 20 bebidas con las 10 menos y 10 más vendidas para poder realizar la comparación entre estas y comparar.

5.1.1.3. Consulta 3 ¿Cuál es la cantidad de deliverys por repartidor del producto más vendido durante el primer semestre del año 2021 a clientes que hayan comprado en locales durante años anteriores? (si más de un producto son los más vendidos mostrar los deliverys para todos).

Justificación: En la empresa es necesario conocer si hay clientes que estén migrando de comprar en locales a pedir deliverys en el primer semestre del año 2021 para determinar si es necesario contratar más repartidores. En este caso, hacemos uso del producto más vendido para conseguir a aquellos repartidores

que entregaron este producto a los clientes que han migrado su preferencia de compra del local a delivery.

5.1.2. Implementación de consultas en SQL

Consulta 1

```

1 select extract(month from c.fecha_hora) as mes, count(c.correlativo
    ) as compras_con_tarjeta_mayores_al_promedio
2 from compra c join compralocal cl on (c.correlativo = cl.
    correlativo) join caja ca on (cl.numero = ca.numero and cl.
    direccion = ca.direccion)
3 where ca.direccion in (
4     select direccion from (select max(total_por_local) as
        maximo_total
5         from (select ca1.direccion, sum(c1.total) as total_por_local
6             from compra c1 join compralocal cl1 on (c1.correlativo = cl1.
                correlativo) join caja ca1 on (cl1.numero = ca1.numero and cl1.
                direccion = ca1.direccion)
7                 group by ca1.direccion) s1) s2
8     join (select ca1.direccion, sum(c1.total) as total_por_local
9         from compra c1 join compralocal cl1 on (c1.correlativo = cl1.
                correlativo) join caja ca1 on (cl1.numero = ca1.numero and cl1.
                direccion = ca1.direccion)
10                group by ca1.direccion) s3 on (maximo_total = total_por_local)
11 )
12 and c.total > (select avg(total) from compra)
13 and (c.metodo_pago = 'Visa' or c.metodo_pago = 'MasterCard')
14 and extract(year from c.fecha_hora) = '2021'
15 group by extract(month from c.fecha_hora)
16 order by mes;
```

Consulta 2

```

1 SELECT nombre_producto, veces FROM ((SELECT nombre_producto, sum(
    cantidad) veces FROM COMPRA as C , (SELECT nombre_producto,
    correlativo, cantidad FROM itemcompra
2 WHERE nombre_producto IN (
3     SELECT nombre_producto FROM entregaproductos
4     WHERE ruc=(SELECT foo.ruc FROM (
5         SELECT ruc, COUNT(nombre_producto) FROM
entregaproductos
6             WHERE medida LIKE '%oz' OR medida LIKE '%L' OR
medida LIKE '%ml'
7                 GROUP BY ruc
8                 ORDER BY 2 DESC
9                 LIMIT 1) AS foo))) AS Item
10                WHERE C.correlativo = Item.correlativo
11                AND (extract(month from fecha_hora) IN ('1','2','3'
12 , '12','11','10'))
13                    AND (extract(year from fecha_hora) IN ('2020','2021
14 '))
15                        GROUP BY nombre_producto
16                        ORDER BY 2
17                        LIMIT 10)
18 UNION
19 (SELECT nombre_producto, sum(cantidad) veces FROM COMPRA as C , (
20     SELECT nombre_producto, correlativo, cantidad FROM itemcompra
```

```

18 WHERE nombre_producto IN (
19     SELECT nombre_producto FROM entregaproductos
20     WHERE ruc=(SELECT foo.ruc FROM (
21         SELECT ruc, COUNT(nombre_producto) FROM
22         entregaproductos
23         WHERE medida LIKE '%oz' OR medida LIKE '%L' OR
24         medida LIKE '%ml'
25         GROUP BY ruc
26         ORDER BY 2 DESC
27         LIMIT 1) AS foo)) AS Item
28     WHERE C.correlativo = Item.correlativo
29     AND (extract(month from fecha_hora) IN ('1','2','3'
30 , '12','11','10'))
31     AND (extract(year from fecha_hora) IN ('2020','2021
32 '))
33             GROUP BY nombre_producto
34             ORDER BY 2 DESC
35             LIMIT 10 ) AS foo1 ORDER BY 2;

```

Consulta 3

```

1
2 SELECT r.dni AS dni_repartidor, nombre AS nombre_repartidor, count(
3     r.dni) AS cantidad_entregas, nombre_producto
4 FROM Delivery d JOIN ItemCompra i ON (d.correlativo = i.correlativo
5 ) JOIN Repartidor r on (d.rdn1 = r.dni) JOIN Persona p2 on (r.
6     dni = p2.dni)
7 WHERE i.nombre_producto IN (
8     SELECT nombre_producto
9     FROM (
10        (SELECT nombre_producto, sum(cantidad) AS cant
11        FROM ItemCompra i JOIN Delivery d ON i.correlativo = d.
12        correlativo JOIN Compra c ON i.correlativo = c.correlativo
13        WHERE
14            EXTRACT(MONTH FROM c.fecha_hora)
15            IN ('1', '2', '3', '4', '5', '6')
16            AND EXTRACT(YEAR FROM c.fecha_hora) = '2021'
17        GROUP BY nombre_producto)
18        T3 JOIN
19        (SELECT max(cant) AS maximo FROM (
20            SELECT sum(cantidad) AS cant
21            FROM ItemCompra i JOIN Delivery d ON i.correlativo = d.
22        correlativo JOIN Compra c ON i.correlativo = c.correlativo
23        WHERE
24            EXTRACT(MONTH FROM c.fecha_hora)
25            IN ('1', '2', '3', '4', '5', '6')
26            AND EXTRACT(YEAR FROM c.fecha_hora) = '2021'
27        GROUP BY nombre_producto
28        ) T2
29        ) T4
30        ON (maximo = cant)
31    ) T1
32 )
33 AND cdni IN (
34     SELECT b.dni
35     FROM CompraLocal cl JOIN Boleta b ON cl.correlativo = b.
36     correlativo JOIN Compra c ON cl.correlativo = c.correlativo
37     WHERE EXTRACT(YEAR FROM c.fecha_hora) < '2021'

```

```
32 )
33 GROUP BY r.dni, nombre, nombre_producto;
```

5.2. Metodología del experimento

Antes de iniciar las pruebas, se crearán 4 esquemas del proyecto con 1000 (1K), 10 000 (10K), 100 000 (100K) y 1 000 000 (1M) de tuplas. Para probar el rendimiento de las 3 consultas planteadas, seguiremos el siguiente procedimiento para cada uno de los esquemas:

5.2.1. Prueba sin índices

Esta prueba se realizará 5 veces y se considerará el promedio de las 5 observaciones para la comparación. Previamente a la realización del procedimiento que sigue, se desactivarán los índices que hemos creado al ejecutar el comando **DROP INDEX**, y se obtendrá el plan de ejecución utilizando la herramienta *pgAdmin 4*.

1. Primero, se ejecutará el comando **VACUUM** en cada una de las tablas, para borrar la caché de PostgreSQL, de forma tal que todas las pruebas se realicen bajo igualdad de condiciones.
2. .
3. En segundo lugar, se ejecutará la consulta utilizando el comando **EXPLAIN ANALYZE**, para probar el rendimiento de la misma.
4. En tercer lugar, se registrarán los resultados obtenidos en términos de tiempo, para analizarlos posteriormente.

5.2.2. Prueba con índices

Esta prueba se realizará 5 veces y se considerará el promedio de las 5 observaciones para la comparación. Previamente a la realización del procedimiento que sigue, se crearán los índices propuestos para optimizar las consultas y se obtendrá el plan de ejecución utilizando la herramienta *pgAdmin 4*.

1. Primero, se ejecutará el comando **VACUUM** en cada una de las tablas, para borrar la caché de PostgreSQL, de forma tal que todas las pruebas se realicen bajo igualdad de condiciones.
2. En segundo lugar, se ejecutará la consulta utilizando el comando **EXPLAIN ANALYZE**, para probar el rendimiento de la misma.
3. En tercer lugar, se registrarán los resultados obtenidos en términos de tiempo, para analizarlos posteriormente.

5.3. Optimización de consultas

A continuación, se crean índices para los atributos que participan con el propósito de evidenciar el impacto que estos tienen en el tiempo de ejecución, es decir, en optimizar el rendimiento de las consultas.

5.3.1. Planes de índices para Consulta 1

```
1 CREATE INDEX index_compralocal_numero_direccion_btree ON
compralocal USING btree(numero, direccion);
```

5.3.1.1. Ejecución sin índices para 1k datos

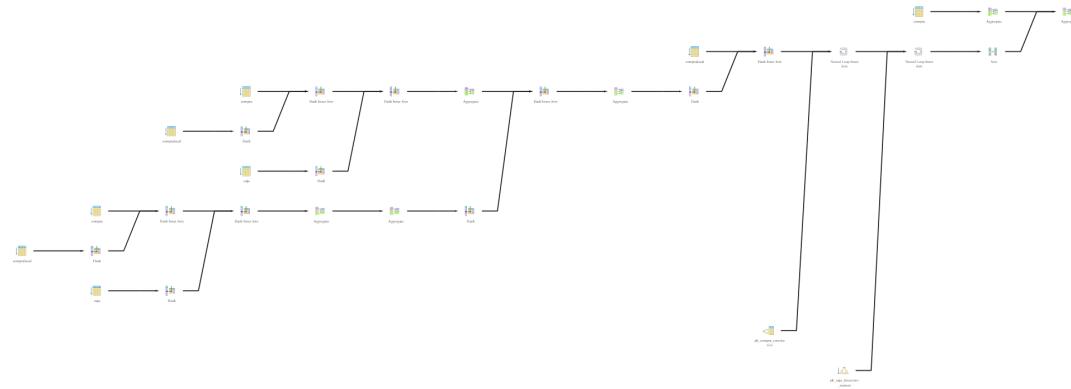


Figura 25: Query Plan Tree de la consulta 1 de 1k datos sin índices

1	GroupAggregate (cost=154.96..154.98 rows=1 width=40) (actual time=2.850..2.862 rows=3 loops=1)
2	[...] Group Key: (EXTRACT(month FROM c.fecha_hora))
3	[...] InitPlan 1 (returns \$0)
4	[...] -> Aggregate (cost=20.50..20.51 rows=1 width=32) (actual time=0.207..0.208 rows=1 loops=1)
5	[...] -> Seq Scan on compra (cost=0.00..18.00 rows=1000 width=5) (actual time=0.004..0.073 rows=1000 loops=1)
6	[...] -> Sort (cost=134.44..134.45 rows=1 width=36) (actual time=2.844..2.853 rows=3 loops=1)
7	[...] Sort Key: (EXTRACT(month FROM c.fecha_hora))
8	[...] Sort Method: quicksort Memory: 25kB
9	[...] -> Nested Loop (cost=96.20..134.43 rows=1 width=36) (actual time=1.984..2.841 rows=3 loops=1)
10	[...] -> Nested Loop (cost=96.06..134.26 rows=1 width=172) (actual time=1.971..2.820 rows=3 loops=1)
11	[...] -> Hash Join (cost=95.79..110.64 rows=60 width=164) (actual time=1.670..1.910 rows=365 loops=1)
12	[...] Hash Cond: ((cl.direccion)=text = (ca1.direccion).text)
13	[...] -> Seq Scan on compralocal cl (cost=0.00..12.00 rows=600 width=46) (actual time=0.008..0.062 rows=600 loops=1)
14	[...] -> Hash (cost=95.78..95.78 rows=1 width=118) (actual time=1.656..1.664 rows=1 loops=1)
15	[...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
16	[...] -> HashAggregate (cost=95.77..95.78 rows=1 width=118) (actual time=1.655..1.663 rows=1 loops=1)
17	[...] Group Key: (ca1.direccion).text
18	[...] Batches: 1 Memory Usage: 24kB
19	[...] -> Hash Join (cost=95.50..95.56 rows=1 width=118) (actual time=1.652..1.660 rows=1 loops=1)
20	[...] Hash Cond: ((sum(c1.total)) = (max((sum(c1..1.total)))))
21	[...] -> HashAggregate (cost=47.61..47.73 rows=10 width=150) (actual time=0.781..0.787 rows=2 loops=1)
22	[...] Group Key: ca1.direccion
23	[...] Batches: 1 Memory Usage: 24kB

```

24 [...] -> Hash Join (cost=20.75..44.61 rows=600 width=123) (actual time=0.150..0.621 rows=600 loops=1)
25 [...] Hash Cond: ((cl1.numero = ca1.numero) AND ((cl1.direccion)::text = (ca1.direccion)::text))
26 [...] -> Hash Join (cost=19.50..40.14 rows=600 width=47) (actual time=0.136..0.442 rows=600 loops=1)
27 [...] Hash Cond: (c1.correlativo = cl1.correlativo)
28 [...] -> Seq Scan on compra c1 (cost=0.00..18.00 rows=1000 width=9) (actual time=0.004..0.087 rows=1000 loops=1)
29 [...] -> Hash (cost=12.00..12.00 rows=600 width=46) (actual time=0.118..0.120 rows=600 loops=1)
30 [...] Buckets: 1024 Batches: 1 Memory Usage: 54kB
31 [...] -> Seq Scan on compralocal cl1 (cost=0.00..12.00 rows=600 width=46) (actual time=0.002..0.047 rows=600 loops=1)
32 [...] -> Hash (cost=1.10..1.10 rows=10 width=120) (actual time=0.009..0.010 rows=10 loops=1)
33 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
34 [...] -> Seq Scan on caja ca1 (cost=0.00..1.10 rows=10 width=120) (actual time=0.002..0.003 rows=10 loops=1)
35 [...] -> Hash (cost=47.88..47.88 rows=1 width=32) (actual time=0.864..0.867 rows=1 loops=1)
36 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
37 [...] -> Aggregate (cost=47.86..47.87 rows=1 width=32) (actual time=0.862..0.864 rows=1 loops=1)
38 [...] -> HashAggregate (cost=47.61..47.73 rows=10 width=150) (actual time=0.857..0.859 rows=2 loops=1)
39 [...] Group Key: ca1_1.direccion
40 [...] Batches: 1 Memory Usage: 24kB
41 [...] -> Hash Join (cost=20.75..44.61 rows=600 width=123) (actual time=0.158..0.644 rows=600 loops=1)
42 [...] Hash Cond: ((cl1_1.numero = ca1_1.numero) AND ((cl1_1.direccion)::text = (ca1_1.direccion)::text))
43 [...] -> Hash Join (cost=19.50..40.14 rows=600 width=47) (actual time=0.136..0.449 rows=600 loops=1)
44 [...] Hash Cond: (c1_1.correlativo = cl1_1.correlativo)
45 [...] -> Seq Scan on compra c1_1 (cost=0.00..18.00 rows=1000 width=9) (actual time=0.004..0.089 rows=1000 loops=1)
46 [...] -> Hash (cost=12.00..12.00 rows=600 width=46) (actual time=0.116..0.117 rows=600 loops=1)

47 [...] Buckets: 1024 Batches: 1 Memory Usage: 54kB
48 [...] -> Seq Scan on compralocal cl1_1 (cost=0.00..12.00 rows=600 width=46) (actual time=0.003..0.045 rows=600 loops=1)
49 [...] -> Hash (cost=1.10..1.10 rows=10 width=120) (actual time=0.010..0.010 rows=10 loops=1)
50 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
51 [...] -> Seq Scan on caja ca1_1 (cost=0.00..1.10 rows=10 width=120) (actual time=0.003..0.004 rows=10 loops=1)
52 [...] -> Index Scan using pk_compra_correlativo on compra c (cost=0.28..0.39 rows=1 width=12) (actual time=0.002..0.002 rows=0 loops=365)
53 [...] Index Cond: (correlativo = cl.correlativo)
54 [...] Filter: ((total > $0) AND (((metodo_pago)::text = 'Visa'::text) OR ((metodo_pago)::text = 'MasterCard'::text)) AND (EXTRACT(year FROM fecha_hora) = '2021'::numeric))
55 [...] Rows Removed by Filter: 1
56 [...] -> Index Only Scan using pk_caja_direccion_numero on caja ca (cost=0.14..0.17 rows=1 width=120) (actual time=0.005..0.005 rows=1 loops=3)
57 [...] Index Cond: ((direccion = (cl.direccion)::text) AND (numero = cl.numero))
58 [...] Heap Fetches: 0
59 Planning Time: 1.055 ms
60 Execution Time: 3.026 ms

```

5.3.1.2. Ejecución con índices para 1k datos

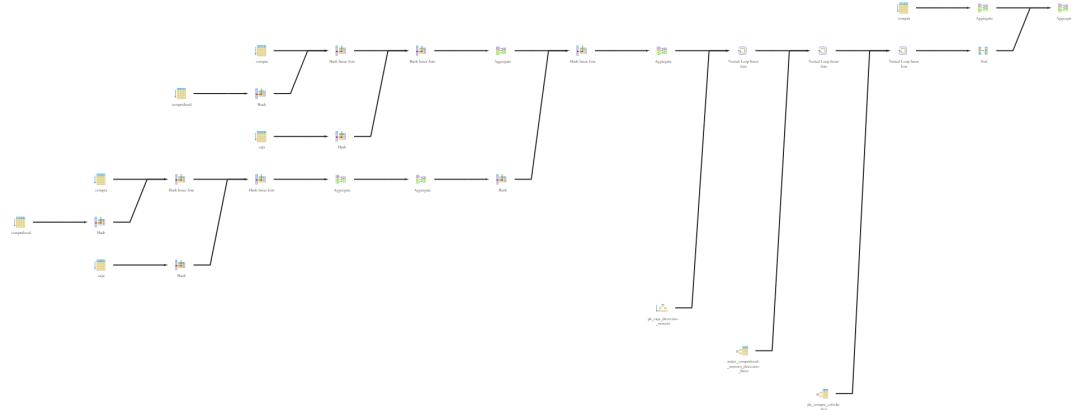


Figura 26: Query Plan Tree de la consulta 1 de 1k datos con índices

5.3.1.3. Ejecución sin índices para 10k datos

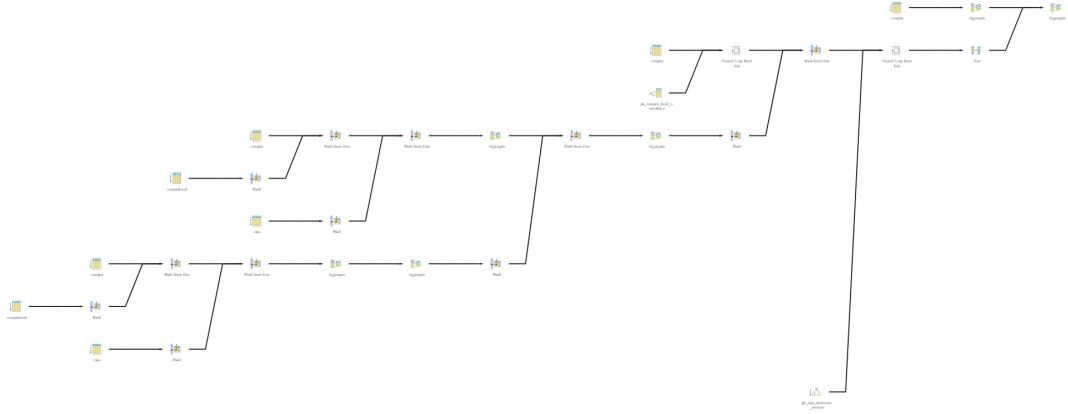


Figura 27: Query Plan Tree de la consulta 1 de 10k datos sin índices

```

1  GroupAggregate (cost=1433.03..1433.06 rows=1 width=40) (actual time=60.723..60.761 rows=12 loops=1)
2  [...] Group Key: (EXTRACT(month FROM c.fechaHora))
3  [...] InitPlan 1 (returns $0)
4  [...] -> Aggregate (cost=195.11..195.12 rows=1 width=32) (actual time=3.902..3.904 rows=1 loops=1)
5  [...] -> Seq Scan on compra (cost=0.00..170.89 rows=9689 width=6) (actual time=0.003..1.458 rows=10000 loops=1)
6  [...] -> Sort (cost=1237.91..1237.91 rows=1 width=36) (actual time=60.711..60.732 rows=49 loops=1)
7  [...] Sort Key: (EXTRACT(month FROM c.fechaHora))
8  [...] Sort Method: quicksort Memory: 27kB
9  [...] -> Nested Loop (cost=904.63..1237.90 rows=1 width=36) (actual time=46.122..60.666 rows=49 loops=1)
10 [...] -> Hash Join (cost=904.49..1237.74 rows=1 width=175) (actual time=46.099..60.506 rows=49 loops=1)
11 [...] Hash Cond: ((cl.direccion)::text = (ca1.direccion)::text)
12 [...] -> Nested Loop (cost=0.28..333.51 rows=3 width=57) (actual time=3.997..18.417 rows=79 loops=1)
13 [...] -> Seq Scan on compra c (cost=0.00..292.00 rows=5 width=12) (actual time=3.962..17.845 rows=127 loops=1)
14 [...] Filter: ((total > $0) AND (((metodo_pago)::text = 'Visa'::text) OR ((metodo_pago)::text = 'MasterCard'::text)) AND (EXTRACT(year FROM fechaHora) = '2021'::numeric))
15 [...] Rows Removed by Filter: 9873
16 [...] -> Index Scan using pk_compra_local_correlativo on compralocal cl (cost=0.28..8.30 rows=1 width=49) (actual time=0.004..0.004 rows=1 loops=127)
17 [...] Index Cond: (correlativo = c.correlativo)
18 [...] -> Hash (cost=904.20..904.20 rows=1 width=118) (actual time=41.988..42.003 rows=1 loops=1)
19 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
20 [...] -> HashAggregate (cost=904.19..904.20 rows=1 width=118) (actual time=41.984..41.998 rows=1 loops=1)
21 [...] Group Key: (ca1.direccion)::text
22 [...] Batches: 1 Memory Usage: 24kB
23 [...] -> Hash Join (cost=903.92..904.18 rows=1 width=118) (actual time=41.977..41.991 rows=1 loops=1)

```

```

24 [...] Hash Cond: ((sum(c1.total)) = (max((sum(c1_1.total)))))
25 [...] > HashAggregate (cost=451.82..451.94 rows=10 width=150) (actual time=27.149..27.158 rows=2 loops=1)
26 [...] Group Key: ca1.direccion
27 [...] Batches: 1 Memory Usage: 24kB
28 [...] > Hash Join (cost=193.25..421.82 rows=6000 width=124) (actual time=5.787..23.632 rows=6000 loops=1)
29 [...] Hash Cond: ((cl1.numero = ca1.numero) AND ((cl1.direccion)::text = (ca1.direccion)::text))
30 [...] > Hash Join (cost=192.00..388.33 rows=6000 width=51) (actual time=5.735..21.129 rows=6000 loops=1)
31 [...] Hash Cond: (c1.correlativo = cl1.correlativo)
32 [...] > Seq Scan on compra c1 (cost=0.00..170.89 rows=9689 width=10) (actual time=0.005..1.253 rows=10000 loops=1)
33 [...] > Hash (cost=117.00..117.00 rows=6000 width=49) (actual time=5.705..5.706 rows=6000 loops=1)
34 [...] Buckets: 8192 Batches: 1 Memory Usage: 541kB
35 [...] > Seq Scan on compralocal cl1 (cost=0.00..117.00 rows=6000 width=49) (actual time=0.004..1.260 rows=6000 loops=1)
36 [...] > Hash (cost=1.10..1.10 rows=10 width=120) (actual time=0.022..0.023 rows=10 loops=1)
37 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
38 [...] > Seq Scan on caja ca1 (cost=0.00..1.10 rows=10 width=120) (actual time=0.014..0.015 rows=10 loops=1)
39 [...] > Hash (cost=452.09..452.09 rows=1 width=32) (actual time=14.797..14.804 rows=1 loops=1)
40 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
41 [...] > Aggregate (cost=452.07..452.08 rows=1 width=32) (actual time=14.788..14.794 rows=1 loops=1)
42 [...] > HashAggregate (cost=451.82..451.94 rows=10 width=150) (actual time=14.778..14.784 rows=2 loops=1)
43 [...] Group Key: ca1_1.direccion
44 [...] Batches: 1 Memory Usage: 24kB
45 [...] > Hash Join (cost=193.25..421.82 rows=6000 width=124) (actual time=1.840..12.474 rows=6000 loops=1)
46 [...] Hash Cond: ((cl1_1.numero = ca1_1.numero) AND ((cl1_1.direccion)::text = (ca1_1.direccion)::text))

```

```

47 [...] > Hash Join (cost=192.00..388.33 rows=6000 width=51) (actual time=1.803..7.927 rows=6000 loops=1)
48 [...] Hash Cond: (c1_1.correlativo = cl1_1.correlativo)
49 [...] > Seq Scan on compra c1_1 (cost=0.00..170.89 rows=9689 width=10) (actual time=0.006..2.072 rows=10000 loops=1)
50 [...] > Hash (cost=117.00..117.00 rows=6000 width=49) (actual time=1.778..1.779 rows=6000 loops=1)
51 [...] Buckets: 8192 Batches: 1 Memory Usage: 541kB
52 [...] > Seq Scan on compralocal cl1_1 (cost=0.00..117.00 rows=6000 width=49) (actual time=0.007..0.656 rows=6000 loops=1)
53 [...] > Hash (cost=1.10..1.10 rows=10 width=120) (actual time=0.023..0.024 rows=10 loops=1)
54 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
55 [...] > Seq Scan on caja ca1_1 (cost=0.00..1.10 rows=10 width=120) (actual time=0.015..0.016 rows=10 loops=1)
56 [...] > Index Only Scan using pk_caja_direccion_numero on caja ca (cost=0.14..0.16 rows=1 width=120) (actual time=0.002..0.002 rows=1 loops=49)
57 [...] Index Cond: ((direccion = (cl.direccion)::text) AND (numero = cl.numero))
58 [...] Heap Fetches: 0
59 Planning Time: 1.691 ms
60 Execution Time: 60.971 ms

```

5.3.1.4. Ejecución con índices para 10k datos

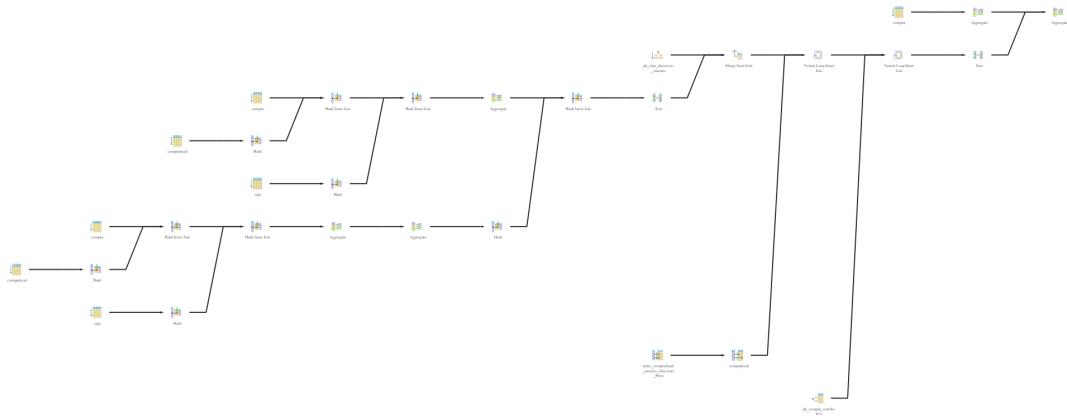


Figura 28: Query Plan Tree de la consulta 1 de 10k datos con índices

5.3.1.5. Ejecución sin índices para 100k datos

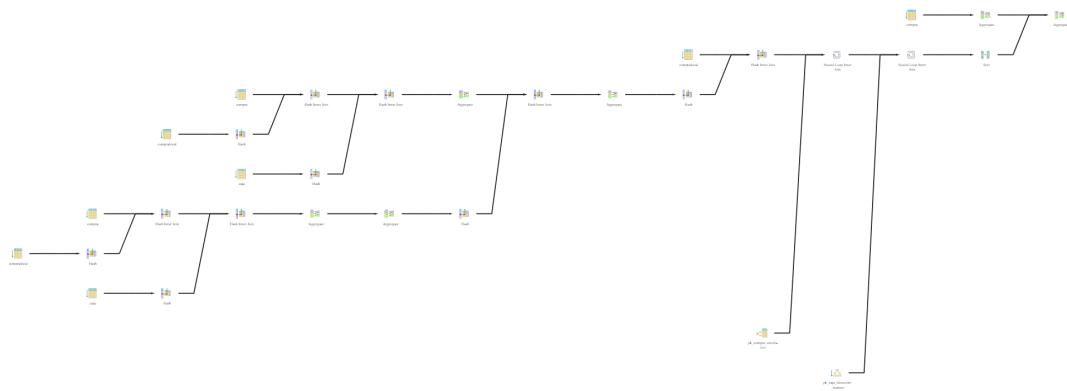


Figura 29: Query Plan Tree de la consulta 1 de 100k datos sin índices

1	GroupAggregate (cost=17588.89..17588.93 rows=2 width=40) (actual time=489.483..489.610 rows=12 loops=1)
2	[...] Group Key: (EXTRACT(month FROM c.fecha_hora))
3	[...] InitPlan 1 (returns \$0)
4	[...] -> Aggregate (cost=1976.52..1976.53 rows=1 width=32) (actual time=24.208..24.210 rows=1 loops=1)
5	[...] -> Seq Scan on compra (cost=0.00..1728.81 rows=99081 width=6) (actual time=0.007..8.804 rows=100000 loops=1)
6	[...] -> Sort (cost=15612.36..15612.37 rows=2 width=36) (actual time=489.462..489.519 rows=320 loops=1)
7	[...] Sort Key: (EXTRACT(month FROM c.fecha_hora))
8	[...] Sort Method: quicksort Memory: 40kB
9	[...] -> Nested Loop (cost=12701.57..15612.35 rows=2 width=36) (actual time=392.930..489.167 rows=320 loops=1)
10	[...] -> Nested Loop (cost=12701.44..15612.04 rows=2 width=163) (actual time=392.899..487.847 rows=320 loops=1)
11	[...] -> Hash Join (cost=12701.14..14051.97 rows=3999 width=155) (actual time=368.529..393.654 rows=27911 loops=1)
12	[...] Hash Cond: ((cl.direccion)::text = (ca1.direccion)::text)
13	[...] -> Seq Scan on compralocal cl (cost=0.00..1085.88 rows=59988 width=37) (actual time=0.013..6.557 rows=60000 loops=1)
14	[...] -> Hash (cost=12701.13..12701.13 rows=1 width=118) (actual time=368.398..368.416 rows=1 loops=1)
15	[...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
16	[...] -> HashAggregate (cost=12701.12..12701.13 rows=1 width=118) (actual time=368.395..368.412 rows=1 loops=1)
17	[...] Group Key: (ca1.direccion)::text
18	[...] Buckets: 1 Memory Usage: 24kB
19	[...] -> Hash Join (cost=12700.73..12701.12 rows=1 width=118) (actual time=368.385..368.406 rows=1 loops=1)
20	[...] Hash Cond: ((sum(c1.total)) = (max((sum(c1_1.total)))))
21	[...] -> HashAggregate (cost=6350.16..6350.35 rows=15 width=150) (actual time=200.746..200.756 rows=3 loops=1)
22	[...] Group Key: ca1.direccion
23	[...] Buckets: 1 Memory Usage: 24kB

24	[...] -> Hash Join (cost=2306.11..6050.22 rows=59988 width=124) (actual time=34.956..178.698 rows=60000 loops=1)
25	[...] Hash Cond: ((cl.numero = ca1.numero) AND ((cl.direccion)::text = (ca1.direccion)::text))
26	[...] -> Hash Join (cost=2304.73..5730.64 rows=59988 width=39) (actual time=34.925..147.525 rows=60000 loops=1)
27	[...] Hash Cond: (c1.correlativo = c1.correlativo)
28	[...] -> Seq Scan on compra cl1 (cost=0.00..1728.81 rows=99081 width=10) (actual time=0.009..23.713 rows=100000 loops=1)
29	[...] -> Hash (cost=1085.88..1085.88 rows=59988 width=37) (actual time=34.773..34.774 rows=60000 loops=1)
30	[...] Buckets: 65536 Batches: 2 Memory Usage: 2533kB
31	[...] -> Seq Scan on compralocal cl1 (cost=0.00..1085.88 rows=59988 width=37) (actual time=0.005..8.432 rows=60000 loops=1)
32	[...] -> Hash (cost=1.15..1.15 rows=15 width=120) (actual time=0.017..0.018 rows=15 loops=1)
33	[...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
34	[...] -> Seq Scan on caja ca1 (cost=0.00..1.15 rows=15 width=120) (actual time=0.005..0.007 rows=15 loops=1)
35	[...] -> Hash (cost=6350.56..6350.56 rows=1 width=32) (actual time=167.608..167.616 rows=1 loops=1)
36	[...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
37	[...] -> Aggregate (cost=6350.54..6350.55 rows=1 width=32) (actual time=167.600..167.607 rows=1 loops=1)
38	[...] -> HashAggregate (cost=6350.16..6350.35 rows=15 width=150) (actual time=167.590..167.598 rows=3 loops=1)
39	[...] Group Key: ca1_1.direccion
40	[...] Buckets: 1 Memory Usage: 24kB
41	[...] -> Hash Join (cost=2306.11..6050.22 rows=59988 width=124) (actual time=25.718..146.689 rows=60000 loops=1)
42	[...] Hash Cond: ((cl1_1.numero = ca1_1.numero) AND ((cl1_1.direccion)::text = (ca1_1.direccion)::text))
43	[...] -> Hash Join (cost=2304.73..5730.64 rows=59988 width=39) (actual time=25.030..123.639 rows=60000 loops=1)
44	[...] Hash Cond: (c1_1.correlativo = c1_1.correlativo)
45	[...] -> Seq Scan on compra cl1_1 (cost=0.00..1728.81 rows=99081 width=10) (actual time=0.008..28.977 rows=100000 loops=1)
46	[...] -> Hash (cost=1085.88..1085.88 rows=59988 width=37) (actual time=24.854..24.855 rows=60000 loops=1)

```

47 [...] Buckets: 65536 Batches: 2 Memory Usage: 253kB
48 [...] -> Seq Scan on compralocal cl1_1 (cost=0.00..1085.88 rows=59988 width=37) (actual time=0.010..7.032 rows=60000 loops=1)
49 [...] -> Hash (cost=1.15..1.15 rows=15 width=120) (actual time=0.037..0.038 rows=15 loops=1)
50 [...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
51 [...] -> Seq Scan on caja ca1_1 (cost=0.00..1.15 rows=15 width=120) (actual time=0.024..0.026 rows=15 loops=1)
52 [...] -> Index Scan using pk_compra_correlativo on compra c (cost=0.29..0.39 rows=1 width=12) (actual time=0.003..0.003 rows=0 loops=27911)
53 [...] Index Cond: (correlativo = cl.correlativo)
54 [...] Filter: ((total > $0) AND (((metodo_pago)::text = 'Visa'::text) OR ((metodo_pago)::text = 'MasterCard'::text)) AND (EXTRACT(year FROM fecha_hora) = '2021'::numeric))
55 [...] Rows Removed by Filter: 1
56 [...] -> Index Only Scan using pk_caja_direccion_numero on caja ca (cost=0.14..0.16 rows=1 width=120) (actual time=0.003..0.003 rows=1 loops=320)
57 [...] Index Cond: ((direccion = (cl.direccion)::text) AND (numero = cl.numero))
58 [...] Heap Fetches: 0
59 Planning Time: 1.577 ms
60 Execution Time: 489.931 ms

```

5.3.1.6. Ejecución con índices para 100k datos

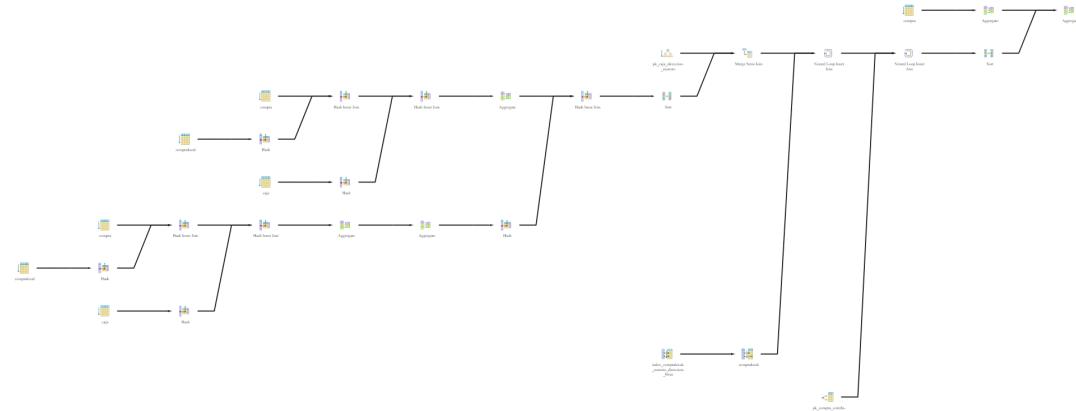


Figura 30: Query Plan Tree de la consulta 1 de 100k datos con índices

5.3.1.7. Ejecución sin índices para 1m datos

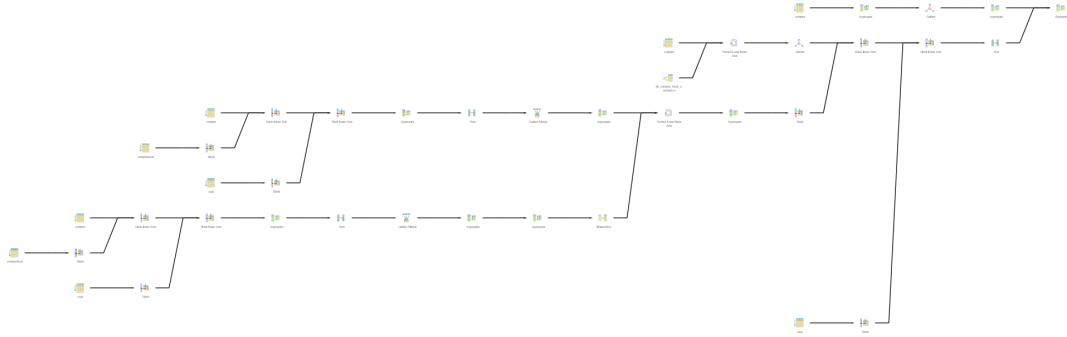


Figura 31: Query Plan Tree de la consulta 1 de 1m datos sin índices

1	GroupAggregate (cost=99097.30..99097.58 rows=14 width=40) (actual time=1602.089..1650.531 rows=12 loops=1)
2	[...] Group Key: (EXTRACT(month FROM c.fechaHora))
3	[...] InitPlan 1 (returns \$1)
4	[...] > Finalize Aggregate (cost=13670.41..13670.42 rows=1 width=32) (actual time=121.043..121.183 rows=1 loops=1)
5	[...] > Gather (cost=13670.19..13670.40 rows=2 width=32) (actual time=121.025..121.168 rows=3 loops=1)
6	[...] Workers Planned: 2
7	[...] Workers Launched: 2
8	[...] > Partial Aggregate (cost=12670.19..12670.20 rows=1 width=32) (actual time=116.840..116.848 rows=1 loops=3)
9	[...] > Parallel Seq Scan on compra (cost=0.00..11631.95 rows=415295 width=6) (actual time=0.060..50.931 rows=333333 loops=3)
10	[...] > Sort (cost=85426.89..85426.92 rows=14 width=36) (actual time=1602.031..1650.041 rows=1574 loops=1)
11	[...] Sort Key: (EXTRACT(month FROM c.fechaHora))
12	[...] Sort Method: quicksort Memory: 122kB
13	[...] > Hash Join (cost=66767.59..85426.62 rows=14 width=36) (actual time=1498.039..1649.020 rows=1574 loops=1)
14	[...] Hash Cond: ((cl.numero = ca.numero) AND ((cl.direccion).text = (ca.direccion).text))
15	[...] > Hash Join (cost=66765.97..85424.89 rows=14 width=149) (actual time=1497.986..1647.831 rows=1574 loops=1)
16	[...] Hash Cond: ((cl.direccion).text = (ca1.direccion).text)
17	[...] > Gather (cost=1000.42..19657.92 rows=342 width=31) (actual time=123.713..228.928 rows=6574 loops=1)
18	[...] Workers Planned: 2
19	[...] Params Evaluated: \$1
20	[...] Workers Launched: 2
21	[...] > Nested Loop (cost=0.42..18623.72 rows=142 width=31) (actual time=0.440..175.792 rows=2191 loops=3)
22	[...] > Parallel Seq Scan on compra c (cost=0.00..16823.13 rows=237 width=12) (actual time=0.327..143.491 rows=3657 loops=3)
23	[...] Filter: ((total > \$1) AND (((metodo_pago).text = 'Visa'.text) OR ((metodo_pago).text = 'MasterCard'.text)) AND (EXTRACT(year FROM fechaHora) = '2021'.numeric))

24	[...] Rows Removed by Filter: 329677
25	[...] > Index Scan using pk_compra_local_correlativo on compralocal cl (cost=0.42..7.60 rows=1 width=23) (actual time=0.008..0.008 rows=1 loops=10970)
26	[...] Index Cond: (correlativo = c.correlativo)
27	[...] > Hash (cost=65765.53..65765.53 rows=1 width=118) (actual time=1374.233..1416.996 rows=1 loops=1)
28	[...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
29	[...] > HashAggregate (cost=65765.52..65765.53 rows=1 width=118) (actual time=1374.227..1416.988 rows=1 loops=1)
30	[...] Group Key: (ca1.direccion):text
31	[...] Batches: 1 Memory Usage: 24kB
32	[...] > Nested Loop (cost=65758.35..65765.52 rows=1 width=118) (actual time=1374.212..1416.980 rows=1 loops=1)
33	[...] Join Filter: ((max((sum(c1_1.total)))) = (sum(c1.total)))
34	[...] Rows Removed by Join Filter: 4
35	[...] > Finalize GroupAggregate (cost=32875.76..32882.28 rows=25 width=150) (actual time=693.846..720.902 rows=5 loops=1)
36	[...] Group Key: ca1.direccion
37	[...] > Gather Merge (cost=32875.76..32881.59 rows=50 width=150) (actual time=693.824..720.856 rows=15 loops=1)
38	[...] Workers Planned: 2
39	[...] Workers Launched: 2
40	[...] > Sort (cost=31875.73..31875.80 rows=25 width=150) (actual time=689.985..689.999 rows=5 loops=3)
41	[...] Sort Key: ca1.direccion
42	[...] Sort Method: quicksort Memory: 25kB
43	[...] Worker 0: Sort Method: quicksort Memory: 25kB
44	[...] Worker 1: Sort Method: quicksort Memory: 25kB
45	[...] > Partial HashAggregate (cost=31874.84..31875.15 rows=25 width=150) (actual time=689.939..689.955 rows=5 loops=3)
46	[...] Group Key: ca1.direccion
47	[...] Batches: 1 Memory Usage: 24kB
48	[...] Worker 0: Batches: 1 Memory Usage: 24kB
49	[...] Worker 1: Batches: 1 Memory Usage: 24kB
50	[...] > Hash Join (cost=11064.75..30625.04 rows=249961 width=124) (actual time=277.292..601.470 rows=200000 loops=3)
51	[...] Hash Cond: ((cl1.numero = ca1.numero) AND ((cl1.direccion):text = (ca1.direccion):text))
52	[...] > Parallel Hash Join (cost=11063.12..29306.22 rows=249961 width=25) (actual time=277.220..512.504 rows=200000 loops=3)
53	[...] Hash Cond: (c1.correlativo = cl1.correlativo)
54	[...] > Parallel Seq Scan on compra c1 (cost=0.00..11631.95 rows=415295 width=10) (actual time=0.018..69.985 rows=333333 loops=3)
55	[...] > Parallel Hash (cost=6473.61..6473.61 rows=249961 width=23) (actual time=96.891..96.897 rows=200000 loops=3)
56	[...] Buckets: 65536 Batches: 16 Memory Usage: 2688kB
57	[...] > Parallel Seq Scan on compralocal cl1 (cost=0.00..6473.61 rows=249961 width=23) (actual time=0.012..35.357 rows=200000 loops=3)
58	[...] > Hash (cost=1.25..1.25 rows=25 width=120) (actual time=0.029..0.030 rows=25 loops=3)
59	[...] Buckets: 1024 Batches: 1 Memory Usage: 10kB
60	[...] > Seq Scan on caja ca1 (cost=0.00..1.25 rows=25 width=120) (actual time=0.017..0.020 rows=25 loops=3)
61	[...] > Materialize (cost=32882.59..32882.62 rows=1 width=32) (actual time=136.059..139.207 rows=1 loops=5)
62	[...] > Aggregate (cost=32882.59..32882.60 rows=1 width=32) (actual time=680.287..696.022 rows=1 loops=1)
63	[...] > Finalize GroupAggregate (cost=32875.76..32882.28 rows=25 width=150) (actual time=680.266..696.012 rows=5 loops=1)
64	[...] Group Key: ca1_1.direccion
65	[...] > Gather Merge (cost=32875.76..32881.59 rows=50 width=150) (actual time=680.224..695.963 rows=15 loops=1)
66	[...] Workers Planned: 2
67	[...] Workers Launched: 2
68	[...] > Sort (cost=31875.73..31875.80 rows=25 width=150) (actual time=674.815..674.971 rows=5 loops=3)
69	[...] Sort Key: ca1_1.direccion

```

70 [...] Sort Method: quicksort Memory: 25kB
71 [...] Worker 0: Sort Method: quicksort Memory: 25kB
72 [...] Worker 1: Sort Method: quicksort Memory: 25kB
73 [...] -> Partial HashAggregate (cost=31874.84..31875.15 rows=25 width=150) (actual time=674.756..674.914 rows=5 loops=3)
74 [...] Group Key: ca1_1.direccion
75 [...] Batches: 1 Memory Usage: 24kB
76 [...] Worker 0: Batches: 1 Memory Usage: 24kB
77 [...] Worker 1: Batches: 1 Memory Usage: 24kB
78 [...] -> Hash Join (cost=11064.75..30625.04 rows=249961 width=124) (actual time=246.541..600.981 rows=200000 loops=3)
79 [...] Hash Cond: ((cl1_1.numero = ca1_1.numero) AND ((cl1_1.direccion)::text = (ca1_1.direccion)::text))
80 [...] -> Parallel Hash Join (cost=11063.12..29306.22 rows=249961 width=25) (actual time=246.475..505.371 rows=200000 loops=3)
81 [...] Hash Cond: (c1_1.correlativo = cl1_1.correlativo)
82 [...] -> Parallel Seq Scan on compra c1_1 (cost=0.00..11631.95 rows=415295 width=10) (actual time=0.031..65.000 rows=333333 loops=3)
83 [...] -> Parallel Hash (cost=6473.61..6473.61 rows=249961 width=23) (actual time=87.325..87.327 rows=200000 loops=3)
84 [...] Buckets: 65536 Batches: 16 Memory Usage: 2688kB
85 [...] -> Parallel Seq Scan on compralocal cl1_1 (cost=0.00..6473.61 rows=249961 width=23) (actual time=0.011..25.637 rows=200000 loops=3)
86 [...] -> Hash (cost=1.25..1.25 rows=25 width=120) (actual time=0.030..0.031 rows=25 loops=3)
87 [...] Buckets: 1024 Batches: 1 Memory Usage: 10kB
88 [...] -> Seq Scan on caja ca1_1 (cost=0.00..1.25 rows=25 width=120) (actual time=0.018..0.021 rows=25 loops=3)
89 [...] -> Hash (cost=1.25..1.25 rows=25 width=120) (actual time=0.026..0.028 rows=25 loops=1)
90 [...] Buckets: 1024 Batches: 1 Memory Usage: 10kB
91 [...] -> Seq Scan on caja ca (cost=0.00..1.25 rows=25 width=120) (actual time=0.014..0.018 rows=25 loops=1)
92 Planning Time: 1.658 ms
93 Execution Time: 1250.000 ms

```

5.3.1.8. Ejecución con índices para 1m datos

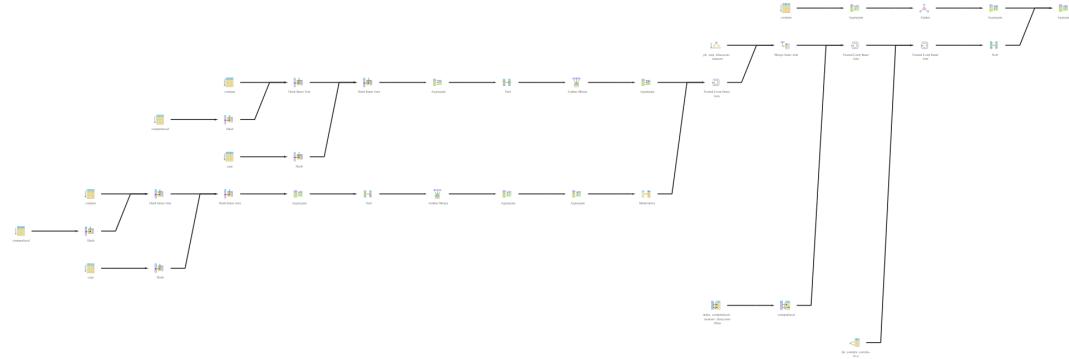


Figura 32: Query Plan Tree de la consulta 1 de 1m datos con índices

5.3.1.9. Explicación de la consulta

Sin índices: Podemos observar que los planes de consulta sin índices no varían mucho. En todos los casos se utilizan los índices para las llaves primarias que PostgreSQL crea por defecto. Se observa que en todos los planes de consulta, se utiliza extensivamente el Hash-Join, y pocas veces un Nested-Loop con índices.

Con índices: Para todos los esquemas, hay una diferencia notoria: el uso de 3 Nested-Loop con índices en lugar de 2. Este Nested-Loop con índices adi-

cional aprovecha el índice btree creado sobre los atributos número y dirección de CompraLocal.

5.3.2. Planes de índices para Consulta 2

```
1 CREATE INDEX index_entrega_productos_ruc ON entregaproductos USING  
hash(ruc);
```

5.3.2.1. Ejecución sin índices para 1k datos

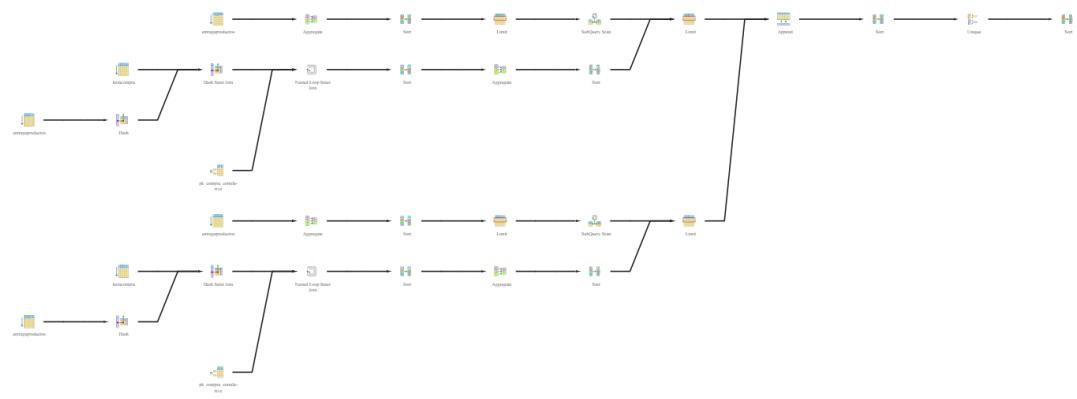


Figura 33: Query Plan Tree de la consulta 2 de 1k datos sin índices

1	Sort (cost=171.17..171.18 rows=2 width=86) (actual time=2.811..2.822 rows=3 loops=1)
2	[...] Sort Key: (sum(itemcompra.cantidad))
3	[...] Sort Method: quicksort Memory: 25kB
4	[...] -> Unique (cost=171.13..171.14 rows=2 width=86) (actual time=2.714..2.727 rows=3 loops=1)
5	[...] -> Sort (cost=171.13..171.13 rows=2 width=86) (actual time=2.713..2.723 rows=6 loops=1)
6	[...] Sort Key: itemcompra.nombre_producto, (sum(itemcompra.cantidad))
7	[...] Sort Method: quicksort Memory: 25kB
8	[...] -> Append (cost=85.54..171.12 rows=2 width=86) (actual time=1.380..2.601 rows=6 loops=1)
9	[...] -> Limit (cost=85.54..85.54 rows=1 width=16) (actual time=1.379..1.386 rows=3 loops=1)
10	[...] InitPlan 1 (returns \$0)
11	[...] -> Subquery Scan on foo (cost=33.46..33.47 rows=1 width=12) (actual time=0.458..0.460 rows=1 loops=1)
12	[...] -> Limit (cost=33.46..33.46 rows=1 width=20) (actual time=0.457..0.459 rows=1 loops=1)
13	[...] -> Sort (cost=33.46..33.69 rows=94 width=20) (actual time=0.456..0.457 rows=1 loops=1)
14	[...] Sort Key: (count(entregaproductos_2.nombre_producto)) DESC
15	[...] Sort Method: top-N heapsort Memory: 25kB
16	[...] -> HashAggregate (cost=32.05..32.99 rows=94 width=20) (actual time=0.398..0.410 rows=80 loops=1)
17	[...] Group Key: entregaproductos_2.ruc

18	[...] Batches: 1 Memory Usage: 24kB
19	[...] -> Seq Scan on entregaproductos entregaproductos_2 (cost=0.00..30.50 rows=310 width=19) (actual time=0.011..0.293 rows=349 loops=1)
20	[...] Filter: (((medida)::text ~~ '%oz'::text) OR ((medida)::text ~~ '%L'::text) OR ((medida)::text ~~ '%ml'::text))
21	[...] Rows Removed by Filter: 651
22	[...] -> Sort (cost=52.07..52.07 rows=1 width=16) (actual time=1.377..1.381 rows=3 loops=1)
23	[...] Sort Key: (sum(itemcompra.cantidad))
24	[...] Sort Method: quicksort Memory: 25kB
25	[...] -> GroupAggregate (cost=52.04..52.06 rows=1 width=16) (actual time=1.344..1.349 rows=3 loops=1)
26	[...] Group Key: itemcompra.nombre_producto
27	[...] -> Sort (cost=52.04..52.04 rows=1 width=12) (actual time=1.339..1.341 rows=3 loops=1)
28	[...] Sort Key: itemcompra.nombre_producto
29	[...] Sort Method: quicksort Memory: 25kB
30	[...] -> Nested Loop (cost=25.90..52.03 rows=1 width=12) (actual time=0.908..1.294 rows=3 loops=1)
31	[...] -> Hash Semi Join (cost=25.62..46.55 rows=15 width=16) (actual time=0.714..1.012 rows=70 loops=1)
32	[...] Hash Cond: ((itemcompra.nombre_producto)::text = (entregaproductos.nombre_producto)::text)
33	[...] -> Seq Scan on itemcompra (cost=0.00..18.00 rows=1000 width=16) (actual time=0.020..0.111 rows=1000 loops=1)
34	[...] -> Hash (cost=25.50..25.50 rows=10 width=7) (actual time=0.621..0.621 rows=71 loops=1)

```

35 [...] Buckets: 1024 Batches: 1 Memory Usage: 11kB
36 [...] -> Seq Scan on entregaproductos (cost=0.00..25.50 rows=10 width=7) (actual time=0.508..0.595 rows=71 loops=1)
37 [...] Filter: ((ruc)::text = ($0)::text)
38 [...] Rows Removed by Filter: 929
39 [...] -> Index Scan using pk_compra_correlativo on compra c (cost=0.28..0.36 rows=1 width=4) (actual time=0.004..0.004 rows=0 loops=70)
40 [...] Index Cond: (correlativo = itemcompra.correlativo)
41 [...] Filter: ((EXTRACT(year FROM fechaHora) = ANY ('(2020,2021)::numeric[])) AND (EXTRACT(month FROM fechaHora) = ANY ('(1,2,3,12,11,10)::numeric[])))
42 [...] Rows Removed by Filter: 1
43 [...] -> Limit (cost=85.54..85.54 rows=1 width=16) (actual time=1.208..1.211 rows=3 loops=1)
44 [...] InitPlan 2 (returns $1)
45 [...] -> Subquery Scan on foo_1 (cost=33.46..33.47 rows=1 width=12) (actual time=0.389..0.390 rows=1 loops=1)
46 [...] -> Limit (cost=33.46..33.46 rows=1 width=20) (actual time=0.388..0.389 rows=1 loops=1)
47 [...] -> Sort (cost=33.46..33.69 rows=94 width=20) (actual time=0.387..0.388 rows=1 loops=1)
48 [...] Sort Key: (count(entregaproductos_3.nombre_producto)) DESC
49 [...] Sort Method: top-N heapsort Memory: 25kB
50 [...] -> HashAggregate (cost=32.05..32.99 rows=94 width=20) (actual time=0.332..0.348 rows=80 loops=1)
51 [...] Group Key: entregaproductos_3.ruc

```

```

52 [...] Batches: 1 Memory Usage: 24kB
53 [...] -> Seq Scan on entregaproductos entregaproductos_3 (cost=0.00..30.50 rows=310 width=19) (actual time=0.004..0.231 rows=349 loops=1)
54 [...] Filter: (((medida)::text ~~ '%oz'::text) OR ((medida)::text ~~ '%L'::text) OR ((medida)::text ~~ '%ml'::text))
55 [...] Rows Removed by Filter: 651
56 [...] -> Sort (cost=52.07..52.07 rows=1 width=16) (actual time=1.208..1.209 rows=3 loops=1)
57 [...] Sort Key: (sum(itemcompra_1.cantidad)) DESC
58 [...] Sort Method: quicksort Memory: 25kB
59 [...] -> GroupAggregate (cost=52.04..52.06 rows=1 width=16) (actual time=1.173..1.175 rows=3 loops=1)
60 [...] Group Key: itemcompra_1.nombre_producto
61 [...] -> Sort (cost=52.04..52.04 rows=1 width=12) (actual time=1.169..1.170 rows=3 loops=1)
62 [...] Sort Key: itemcompra_1.nombre_producto
63 [...] Sort Method: quicksort Memory: 25kB
64 [...] -> Nested Loop (cost=25.90..52.03 rows=1 width=12) (actual time=0.785..1.140 rows=3 loops=1)
65 [...] -> Hash Semi Join (cost=25.62..46.55 rows=15 width=16) (actual time=0.647..0.918 rows=70 loops=1)
66 [...] Hash Cond: ((itemcompra_1.nombre_producto)::text = (entregaproductos_1.nombre_producto)::text)
67 [...] -> Seq Scan on itemcompra itemcompra_1 (cost=0.00..18.00 rows=1000 width=16) (actual time=0.004..0.097 rows=1000 loops=1)
68 [...] -> Hash (cost=25.50..25.50 rows=10 width=7) (actual time=0.600..0.600 rows=71 loops=1)

```

```

69 [...] Buckets: 1024 Batches: 1 Memory Usage: 11kB
70 [...]-> Seq Scan on entregaproductos entregaproductos_1 (cost=0.00..25.50 rows=10 width=7) (actual time=0.435..0.550 rows=71 loops=1)
71 [...] Filter: ((ruc)::text = ($1)::text)
72 [...] Rows Removed by Filter: 929
73 [...]-> Index Scan using pk_compra_correlativo on compra c_1 (cost=0.28..0.36 rows=1 width=4) (actual time=0.003..0.003 rows=0 loops=70)
74 [...] Index Cond: (correlativo = itemcompra_1.correlativo)
75 [...] Filter: ((EXTRACT(year FROM fechaHora) = ANY ('(2020,2021)')::numeric[])) AND ((EXTRACT(month FROM fechaHora) = ANY ('{1,2,3,12,11,10}')::numeric[]))
76 [...] Rows Removed by Filter: 1
77 Planning Time: 16.365 ms
78 Execution Time: 3.542 ms

```

Figura 34: Query Plan de la consulta 2 de 1k datos sin índices

5.3.2.2. Ejecución con índices para 1k datos

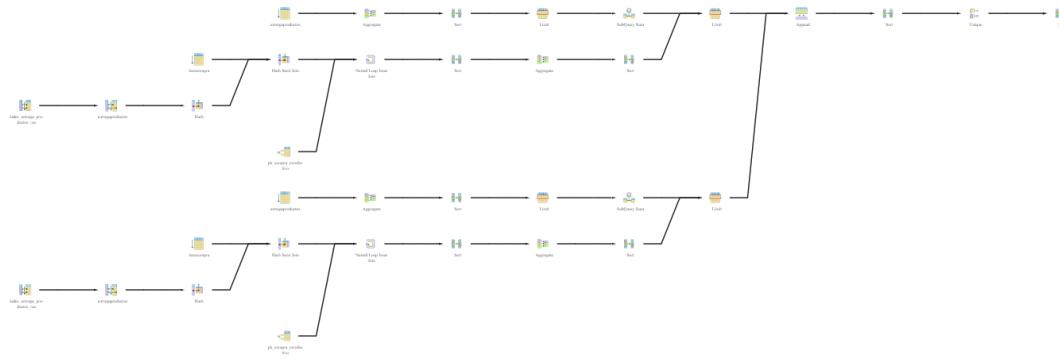


Figura 35: Query Plan Tree de la consulta 2 de 1k datos con índices

1	Sort (cost=154.92..154.93 rows=2 width=86) (actual time=2.760..2.769 rows=3 loops=1)
2	[...] Sort Key: (sum(itemcompra.cantidad))
3	[...] Sort Method: quicksort Memory: 25kB
4	[...]-> Unique (cost=154.88..154.89 rows=2 width=86) (actual time=2.717..2.728 rows=3 loops=1)
5	[...]-> Sort (cost=154.88..154.88 rows=2 width=86) (actual time=2.716..2.724 rows=6 loops=1)
6	[...] Sort Key: itemcompra.nombre_producto, (sum(itemcompra.cantidad))
7	[...] Sort Method: quicksort Memory: 25kB
8	[...]-> Append (cost=77.41..154.87 rows=2 width=86) (actual time=1.350..2.702 rows=6 loops=1)
9	[...]-> Limit (cost=77.41..77.42 rows=1 width=16) (actual time=1.349..1.354 rows=3 loops=1)
10	[...] InitPlan 1 (returns \$0)
11	[...]-> Subquery Scan on foo (cost=33.46..33.47 rows=1 width=12) (actual time=0.521..0.523 rows=1 loops=1)
12	[...]-> Limit (cost=33.46..33.46 rows=1 width=20) (actual time=0.520..0.522 rows=1 loops=1)
13	[...]-> Sort (cost=33.46..33.69 rows=94 width=20) (actual time=0.520..0.521 rows=1 loops=1)
14	[...] Sort Key: (count(entregaproductos_2.nombre_producto)) DESC
15	[...] Sort Method: top-N heapsort Memory: 25kB
16	[...]-> HashAggregate (cost=32.05..32.99 rows=94 width=20) (actual time=0.464..0.481 rows=80 loops=1)
17	[...] Group Key: entregaproductos_2.ruc

18	[...] Batches: 1 Memory Usage: 24kB
19	[...]-> Seq Scan on entregaproductos entregaproductos_2 (cost=0.00..30.50 rows=310 width=19) (actual time=0.012..0.336 rows=349 loops=1)
20	[...] Filter: (((medida)::text ~~ '%oz'::text) OR ((medida)::text ~~ '%L'::text) OR ((medida)::text ~~ '%ml'::text))
21	[...] Rows Removed by Filter: 651
22	[...]-> Sort (cost=43.94..43.95 rows=1 width=16) (actual time=1.347..1.351 rows=3 loops=1)
23	[...] Sort Key: (sum(itemcompra.cantidad))
24	[...] Sort Method: quicksort Memory: 25kB
25	[...]-> GroupAggregate (cost=43.91..43.93 rows=1 width=16) (actual time=1.326..1.330 rows=3 loops=1)
26	[...] Group Key: itemcompra.nombre_producto
27	[...]-> Sort (cost=43.91..43.92 rows=1 width=12) (actual time=1.322..1.324 rows=3 loops=1)
28	[...] Sort Key: itemcompra.nombre_producto
29	[...] Sort Method: quicksort Memory: 25kB
30	[...]-> Nested Loop (cost=17.78..43.90 rows=1 width=12) (actual time=0.893..1.285 rows=3 loops=1)
31	[...]-> Hash Semi Join (cost=17.50..38.42 rows=15 width=16) (actual time=0.700..0.982 rows=70 loops=1)
32	[...] Hash Cond: ((itemcompra.nombre_producto)::text = (entregaproductos.nombre_producto)::text)
33	[...]-> Seq Scan on itemcompra (cost=0.00..18.00 rows=1000 width=16) (actual time=0.013..0.115 rows=1000 loops=1)
34	[...]-> Hash (cost=17.38..17.38 rows=10 width=7) (actual time=0.647..0.647 rows=71 loops=1)

35	[...] Buckets: 1024 Batches: 1 Memory Usage: 11kB
36	[...]-> Bitmap Heap Scan on entregaproductos (cost=4.08..17.38 rows=10 width=7) (actual time=0.595..0.616 rows=71 loops=1)
37	[...] Recheck Cond: ((ruc)::text = (\$0)::text)
38	[...] Heap Blocks: exact=4
39	[...]-> Bitmap Index Scan on index_entrega_productos_ruc (cost=0.00..4.08 rows=10 width=0) (actual time=0.590..0.590 rows=71 loops=1)
40	[...] Index Cond: ((ruc)::text = (\$0)::text)
41	[...]-> Index Scan using pk_compra_correlativo on compra c (cost=0.28..0.36 rows=1 width=4) (actual time=0.004..0.004 rows=0 loops=70)
42	[...] Index Cond: (correlativo = itemcompra.correlativo)
43	[...] Filter: ((EXTRACT(year FROM fechaHora) = ANY ('(2020,2021)::numeric[])) AND (EXTRACT(month FROM fechaHora) = ANY ('(1,2,3,12,11,10)::numeric[])))
44	[...] Rows Removed by Filter: 1
45	[...]-> Limit (cost=77.41..77.42 rows=1 width=16) (actual time=1.342..1.345 rows=3 loops=1)
46	[...] InitPlan 2 (returns \$1)
47	[...]-> Subquery Scan on foo_1 (cost=33.46..33.47 rows=1 width=12) (actual time=0.620..0.621 rows=1 loops=1)
48	[...]-> Limit (cost=33.46..33.46 rows=1 width=20) (actual time=0.619..0.619 rows=1 loops=1)
49	[...]-> Sort (cost=33.46..33.69 rows=94 width=20) (actual time=0.618..0.618 rows=1 loops=1)
50	[...] Sort Key: (count(entregaproductos_3.nombre_producto)) DESC
51	[...] Sort Method: top-N heapsort Memory: 25kB
52	[...]-> HashAggregate (cost=32.05..32.99 rows=94 width=20) (actual time=0.513..0.583 rows=80 loops=1)
53	[...] Group Key: entregaproductos_3.ruc
54	[...] Batches: 1 Memory Usage: 24kB
55	[...]-> Seq Scan on entregaproductos entregaproductos_3 (cost=0.00..30.50 rows=310 width=19) (actual time=0.006..0.381 rows=349 loops=1)
56	[...] Filter: (((medida)::text ~~ '%oz'::text) OR ((medida)::text ~~ '%L'::text) OR ((medida)::text ~~ '%ml'::text))
57	[...] Rows Removed by Filter: 651
58	[...]-> Sort (cost=43.94..43.95 rows=1 width=16) (actual time=1.341..1.343 rows=3 loops=1)
59	[...] Sort Key: (sum(itemcompra_1.cantidad)) DESC
60	[...] Sort Method: quicksort Memory: 25kB
61	[...]-> GroupAggregate (cost=43.91..43.93 rows=1 width=16) (actual time=1.322..1.325 rows=3 loops=1)
62	[...] Group Key: itemcompra_1.nombre_producto
63	[...]-> Sort (cost=43.91..43.92 rows=1 width=12) (actual time=1.318..1.319 rows=3 loops=1)
64	[...] Sort Key: itemcompra_1.nombre_producto
65	[...] Sort Method: quicksort Memory: 25kB
66	[...]-> Nested Loop (cost=17.78..43.90 rows=1 width=12) (actual time=0.859..1.293 rows=3 loops=1)
67	[...]-> Hash Semi Join (cost=17.50..38.42 rows=15 width=16) (actual time=0.739..1.010 rows=70 loops=1)
68	[...] Hash Cond: ((itemcompra_1.nombre_producto)::text = (entregaproductos_1.nombre_producto)::text)

```

69 [...]-> Seq Scan on itemcompra itemcompra_1 (cost=0.00..18.00 rows=1000 width=16) (actual time=0.005..0.102 rows=1000 loops=1)
70 [...]-> Hash (cost=17.38..17.38 rows=10 width=7) (actual time=0.702..0.703 rows=71 loops=1)
71 [...] Buckets: 1024 Batches: 1 Memory Usage: 11kB
72 [...]-> Bitmap Heap Scan on entregaproductos entregaproductos_1 (cost=4.08..17.38 rows=10 width=7) (actual time=0.656..0.676 rows=71 loops=1)
73 [...] Recheck Cond: ((ruc)::text = ($1)::text)
74 [...] Heap Blocks: exact=4
75 [...]-> Bitmap Index Scan on index_entrega_productos_ruc (cost=0.00..4.08 rows=10 width=0) (actual time=0.650..0.650 rows=71 loops=1)
76 [...] Index Cond: ((ruc)::text = ($1)::text)
77 [...]-> Index Scan using pk_compra_correlativo on compra c_1 (cost=0.28..0.36 rows=1 width=4) (actual time=0.004..0.004 rows=0 loops=70)
78 [...] Index Cond: (correlativo = itemcompra_1.correlativo)
79 [...] Filter: ((EXTRACT(year FROM fechaHora) = ANY ('(2020,2021)::numeric[])) AND (EXTRACT(month FROM fechaHora) = ANY ('{1,2,3,12,11,10}::numeric[])))
80 [...] Rows Removed by Filter: 1
81 Planning Time: 5.240 ms
82 Execution Time: 3.987 ms

```

Figura 36: Query Plan de la consulta 2 de 1k datos con índices

5.3.2.3. Ejecución sin índices para 10k datos

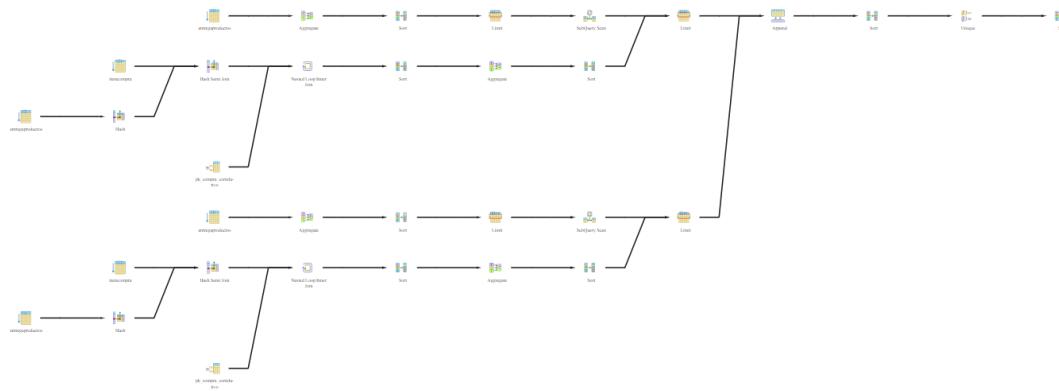


Figura 37: Query Plan Tree de la consulta 2 de 10k datos sin índices

1	Sort (cost=1554.50..1554.51 rows=2 width=86) (actual time=38.852..38.872 rows=20 loops=1)
2	[...] Sort Key: (sum(itemcompra.cantidad))
3	[...] Sort Method: quicksort Memory: 26kB
4	[...] -> Unique (cost=1554.46..1554.47 rows=2 width=86) (actual time=38.823..38.850 rows=20 loops=1)
5	[...] -> Sort (cost=1554.46..1554.46 rows=2 width=86) (actual time=38.822..38.841 rows=20 loops=1)
6	[...] Sort Key: itemcompra.nombre_producto, (sum(itemcompra.cantidad))
7	[...] Sort Method: quicksort Memory: 26kB
8	[...] -> Append (cost=777.20..1554.45 rows=2 width=86) (actual time=28.246..38.809 rows=20 loops=1)
9	[...] -> Limit (cost=777.20..777.21 rows=1 width=15) (actual time=28.244..28.257 rows=10 loops=1)
10	[...] InitPlan 1 (returns \$0)
11	[...] -> Subquery Scan on foo (cost=311.23..311.24 rows=1 width=12) (actual time=5.650..5.655 rows=1 loops=1)
12	[...] -> Limit (cost=311.23..311.23 rows=1 width=20) (actual time=5.649..5.652 rows=1 loops=1)
13	[...] -> Sort (cost=311.23..311.48 rows=100 width=20) (actual time=5.648..5.649 rows=1 loops=1)
14	[...] Sort Key: (count(entregaproductos_2.nombre_producto)) DESC
15	[...] Sort Method: top-N heapsort Memory: 25kB
16	[...] -> HashAggregate (cost=309.73..310.73 rows=100 width=20) (actual time=5.599..5.614 rows=99 loops=1)
17	[...] Group Key: entregaproductos_2.ruc

18	[...] Batches: 1 Memory Usage: 24kB
19	[...] -> Seq Scan on entregaproductos entregaproductos_2 (cost=0.00..294.64 rows=3018 width=19) (actual time=0.009..4.769 rows=3390 loops=1)
20	[...] Filter: (((medida)::text ~~~ '%oz'::text) OR ((medida)::text ~~~ '%L'::text) OR ((medida)::text ~~~ '%ml'::text))
21	[...] Rows Removed by Filter: 6610
22	[...] -> Sort (cost=465.96..465.97 rows=1 width=15) (actual time=28.243..28.249 rows=10 loops=1)
23	[...] Sort Key: (sum(itemcompra.cantidad))
24	[...] Sort Method: top-N heapsort Memory: 26kB
25	[...] -> GroupAggregate (cost=465.93..465.95 rows=1 width=15) (actual time=28.208..28.229 rows=38 loops=1)
26	[...] Group Key: itemcompra.nombre_producto
27	[...] -> Sort (cost=465.93..465.94 rows=1 width=11) (actual time=28.198..28.204 rows=38 loops=1)
28	[...] Sort Key: itemcompra.nombre_producto
29	[...] Sort Method: quicksort Memory: 27kB
30	[...] -> Nested Loop (cost=248.78..465.92 rows=1 width=11) (actual time=14.970..28.137 rows=38 loops=1)
31	[...] -> Hash Semi Join (cost=248.50..421.83 rows=118 width=15) (actual time=8.467..12.721 rows=486 loops=1)
32	[...] Hash Cond: ((itemcompra.nombre_producto)::text = (entregaproductos.nombre_producto)::text)
33	[...] -> Seq Scan on itemcompra (cost=0.00..151.56 rows=7456 width=15) (actual time=0.586..3.088 rows=10000 loops=1)
34	[...] -> Hash (cost=247.31..247.31 rows=95 width=7) (actual time=7.837..7.838 rows=473 loops=1)

35	[...] Buckets: 1024 Batches: 1 Memory Usage: 27kB
36	[...]-> Seq Scan on entregaproductos (cost=0.00..247.31 rows=95 width=7) (actual time=6.124..7.686 rows=473 loops=1)
37	[...] Filter: ((ruc)::text = (\$0)::text)
38	[...] Rows Removed by Filter: 9527
39	[...]-> Index Scan using pk_compra_correlativo on compra c (cost=0.29..0.37 rows=1 width=4) (actual time=0.031..0.031 rows=0 loops=486)
40	[...] Index Cond: (correlativo = itemcompra.correlativo)
41	[...] Filter: ((EXTRACT(year FROM fecha_hora) = ANY ('(2020,2021)::numeric[])) AND (EXTRACT(month FROM fecha_hora) = ANY ('1,2,3,12,11,10)::numeric[])))
42	[...] Rows Removed by Filter: 1
43	[...]-> Limit (cost=777.20..777.21 rows=1 width=15) (actual time=10.536..10.544 rows=10 loops=1)
44	[...] InitPlan 2 (returns \$1)
45	[...]-> Subquery Scan on foo_1 (cost=311.23..311.24 rows=1 width=12) (actual time=4.268..4.273 rows=1 loops=1)
46	[...]-> Limit (cost=311.23..311.23 rows=1 width=20) (actual time=4.267..4.270 rows=1 loops=1)
47	[...]-> Sort (cost=311.23..311.48 rows=100 width=20) (actual time=4.265..4.267 rows=1 loops=1)
48	[...] Sort Key: (count(entregaproductos_3.nombre_producto)) DESC
49	[...] Sort Method: top-N heapsort Memory: 25kB
50	[...]-> HashAggregate (cost=309.73..310.73 rows=100 width=20) (actual time=4.201..4.222 rows=99 loops=1)
51	[...] Group Key: entregaproductos_3.ruc

52	[...] Batches: 1 Memory Usage: 24kB
53	[...]-> Seq Scan on entregaproductos entregaproductos_3 (cost=0.00..294.64 rows=3018 width=19) (actual time=0.011..3.096 rows=3390 loops=1)
54	[...] Filter: (((medida)::text ~~ '%oz'::text) OR ((medida)::text ~~ '%L'::text) OR ((medida)::text ~~ '%ml'::text))
55	[...] Rows Removed by Filter: 6610
56	[...]-> Sort (cost=465.96..465.97 rows=1 width=15) (actual time=10.534..10.537 rows=10 loops=1)
57	[...] Sort Key: (sum(itemcompra_1.cantidad)) DESC
58	[...] Sort Method: top-N heapsort Memory: 26kB
59	[...]-> GroupAggregate (cost=465.93..465.95 rows=1 width=15) (actual time=10.433..10.460 rows=38 loops=1)
60	[...] Group Key: itemcompra_1.nombre_producto
61	[...]-> Sort (cost=465.93..465.94 rows=1 width=11) (actual time=10.422..10.426 rows=38 loops=1)
62	[...] Sort Key: itemcompra_1.nombre_producto
63	[...] Sort Method: quicksort Memory: 27kB
64	[...]-> Nested Loop (cost=248.78..465.92 rows=1 width=11) (actual time=6.292..10.334 rows=38 loops=1)
65	[...]-> Hash Semi Join (cost=248.50..421.83 rows=118 width=15) (actual time=6.040..8.740 rows=486 loops=1)
66	[...] Hash Cond: ((itemcompra_1.nombre_producto)::text = (entregaproductos_1.nombre_producto)::text)
67	[...]-> Seq Scan on itemcompra itemcompra_1 (cost=0.00..151.56 rows=7456 width=15) (actual time=0.006..0.911 rows=10000 loops=1)
68	[...]-> Hash (cost=247.31..247.31 rows=95 width=7) (actual time=5.986..5.986 rows=473 loops=1)

69	[...] Buckets: 1024 Batches: 1 Memory Usage: 27kB
70	[...]-> Seq Scan on entregaproductos entregaproductos_1 (cost=0.00..247.31 rows=95 width=7) (actual time=4.364..5.816 rows=473 loops=1)
71	[...] Filter: ((ruc)::text = (\$1)::text)
72	[...] Rows Removed by Filter: 9527
73	[...]-> Index Scan using pk_compra_correlativo on compra c_1 (cost=0.29..0.37 rows=1 width=4) (actual time=0.003..0.003 rows=0 loops=486)
74	[...] Index Cond: (correlativo = itemcompra_1.correlativo)
75	[...] Filter: ((EXTRACT(year FROM fecha_hora) = ANY ('{2020,2021}'::numeric[])) AND (EXTRACT(month FROM fecha_hora) = ANY ('{1,2,3,12,11,10}'::numeric[])))
76	[...] Rows Removed by Filter: 1
77	Planning Time: 11.027 ms
78	Execution Time: 39.308 ms

Figura 38: Query Plan de la consulta 2 de 10k datos sin índices

5.3.2.4. Ejecución con índices para 10k datos



Figura 39: Query Plan Tree de la consulta 2 de 10k datos con índices

1	Sort (cost=1343.43..1343.43 rows=2 width=86) (actual time=16.411..16.429 rows=20 loops=1)
2	[...] Sort Key: (sum(itemcompra.cantidad))
3	[...] Sort Method: quicksort Memory: 26kB
4	[...]-> Unique (cost=1343.38..1343.40 rows=2 width=86) (actual time=16.391..16.415 rows=20 loops=1)
5	[...]-> Sort (cost=1343.38..1343.39 rows=2 width=86) (actual time=16.390..16.407 rows=20 loops=1)
6	[...] Sort Key: itemcompra.nombre_producto, (sum(itemcompra.cantidad))
7	[...] Sort Method: quicksort Memory: 26kB
8	[...]-> Append (cost=671.67..1343.37 rows=2 width=86) (actual time=9.322..16.383 rows=20 loops=1)
9	[...]-> Limit (cost=671.67..671.67 rows=1 width=15) (actual time=9.321..9.334 rows=10 loops=1)
10	[...] InitPlan 1 (returns \$0)
11	[...]-> Subquery Scan on foo (cost=320.91..320.92 rows=1 width=12) (actual time=4.159..4.163 rows=1 loops=1)
12	[...]-> Limit (cost=320.91..320.91 rows=1 width=20) (actual time=4.159..4.161 rows=1 loops=1)
13	[...]-> Sort (cost=320.91..321.16 rows=100 width=20) (actual time=4.157..4.159 rows=1 loops=1)
14	[...] Sort Key: (count(entregaproductos_2.nombre_producto)) DESC
15	[...] Sort Method: top-N heapsort Memory: 25kB
16	[...]-> HashAggregate (cost=319.41..320.41 rows=100 width=20) (actual time=4.101..4.116 rows=99 loops=1)
17	[...] Group Key: entregaproductos_2.ruc

18	[...] Batches: 1 Memory Usage: 24kB
19	[...]-> Seq Scan on entregaproductos entregaproductos_2 (cost=0.00..303.51 rows=3180 width=19) (actual time=0.016..3.312 rows=3390 loops=1)
20	[...] Filter: (((medida)::text ~~ '%oz'::text) OR ((medida)::text ~~ '%L'::text) OR ((medida)::text ~~ '%mi'::text))
21	[...] Rows Removed by Filter: 6610
22	[...]-> Sort (cost=350.74..350.75 rows=1 width=15) (actual time=9.320..9.327 rows=10 loops=1)
23	[...] Sort Key: (sum(itemcompra.cantidad))
24	[...] Sort Method: top-N heapsort Memory: 26kB
25	[...]-> GroupAggregate (cost=350.71..350.73 rows=1 width=15) (actual time=9.288..9.310 rows=38 loops=1)
26	[...] Group Key: itemcompra.nombre_producto
27	[...]-> Sort (cost=350.71..350.72 rows=1 width=11) (actual time=9.280..9.287 rows=38 loops=1)
28	[...] Sort Key: itemcompra.nombre_producto
29	[...] Sort Method: quicksort Memory: 27kB
30	[...]-> Nested Loop (cost=134.82..350.70 rows=1 width=11) (actual time=4.886..9.237 rows=38 loops=1)
31	[...]-> Hash Semi Join (cost=134.53..306.92 rows=117 width=15) (actual time=4.519..7.176 rows=486 loops=1)
32	[...] Hash Cond: ((itemcompra.nombre_producto)::text = (entregaproductos.nombre_producto)::text)
33	[...]-> Seq Scan on itemcompra (cost=0.00..150.83 rows=7383 width=15) (actual time=0.010..0.788 rows=10000 loops=1)
34	[...]-> Hash (cost=133.28..133.28 rows=100 width=7) (actual time=4.475..4.477 rows=473 loops=1)

35	[...] Buckets: 1024 Batches: 1 Memory Usage: 27kB
36	[...] -> Bitmap Heap Scan on entregaproductos (cost=4.78..133.28 rows=100 width=7) (actual time=4.264..4.365 rows=473 loops=1)
37	[...] Recheck Cond: ((ruc)::text = (\$0)::text)
38	[...] Heap Blocks: exact=10
39	[...] -> Bitmap Index Scan on index_entrega_productos_ruc (cost=0.00..4.75 rows=100 width=0) (actual time=4.255..4.256 rows=473 loops=1)
40	[...] Index Cond: ((ruc)::text = (\$0)::text)
41	[...] -> Index Scan using pk_compra_correlativo on compra c (cost=0.29..0.37 rows=1 width=4) (actual time=0.004..0.004 rows=0 loops=486)
42	[...] Index Cond: (correlativo = itemcompra.correlativo)
43	[...] Filter: ((EXTRACT(year FROM fechaHora) = ANY ('(2020,2021)::numeric[])) AND (EXTRACT(month FROM fechaHora) = ANY ('(1,2,3,12,11,10)::numeric[])))
44	[...] Rows Removed by Filter: 1
45	[...] -> Limit (cost=671.67..671.67 rows=1 width=15) (actual time=7.036..7.042 rows=10 loops=1)
46	[...] InitPlan 2 (returns \$1)
47	[...] -> Subquery Scan on foo_1 (cost=320.91..320.92 rows=1 width=12) (actual time=3.228..3.231 rows=1 loops=1)
48	[...] -> Limit (cost=320.91..320.91 rows=1 width=20) (actual time=3.227..3.229 rows=1 loops=1)
49	[...] -> Sort (cost=320.91..321.16 rows=100 width=20) (actual time=3.226..3.226 rows=1 loops=1)
50	[...] Sort Key: (count(entregaproductos_3.nombre_producto)) DESC
51	[...] Sort Method: top-N heapsort Memory: 25kB
52	[...] -> HashAggregate (cost=319.41..320.41 rows=100 width=20) (actual time=3.184..3.199 rows=99 loops=1)
53	[...] Group Key: entregaproductos_3.ruc
54	[...] Batches: 1 Memory Usage: 24kB
55	[...] -> Seq Scan on entregaproductos entregaproductos_3 (cost=0.00..303.51 rows=3180 width=19) (actual time=0.006..2.368 rows=3390 loops=1)
56	[...] Filter: (((medida)::text ~~ '%oz'::text) OR ((medida)::text ~~ '%L'::text) OR ((medida)::text ~~ '%ml'::text))
57	[...] Rows Removed by Filter: 6610
58	[...] -> Sort (cost=350.74..350.75 rows=1 width=15) (actual time=7.034..7.037 rows=10 loops=1)
59	[...] Sort Key: (sum(itemcompra_1.cantidad)) DESC
60	[...] Sort Method: top-N heapsort Memory: 26kB
61	[...] -> GroupAggregate (cost=350.71..350.73 rows=1 width=15) (actual time=6.982..7.006 rows=38 loops=1)
62	[...] Group Key: itemcompra_1.nombre_producto
63	[...] -> Sort (cost=350.71..350.72 rows=1 width=11) (actual time=6.972..6.977 rows=38 loops=1)
64	[...] Sort Key: itemcompra_1.nombre_producto
65	[...] Sort Method: quicksort Memory: 27kB
66	[...] -> Nested Loop (cost=134.82..350.70 rows=1 width=11) (actual time=3.642..6.921 rows=38 loops=1)
67	[...] -> Hash Semi Join (cost=134.53..306.92 rows=117 width=15) (actual time=3.498..5.642 rows=486 loops=1)
68	[...] Hash Cond: ((itemcompra_1.nombre_producto)::text = (entregaproductos_1.nombre_producto)::text)

```

69 [...] > Seq Scan on itemcompra itemcompra_1 (cost=0.00..150.83 rows=7383 width=15) (actual time=0.010..0.786 rows=10000 loops=1)
70 [...] > Hash (cost=133.28..133.28 rows=100 width=7) (actual time=3.458..3.459 rows=473 loops=1)
71 [...] Buckets: 1024 Batches: 1 Memory Usage: 27kB
72 [...] > Bitmap Heap Scan on entregaproductos entregaproductos_1 (cost=4.78..133.28 rows=100 width=7) (actual time=3.293..3.375 rows=473 loops=1)
73 [...] Recheck Cond: ((ruc).text = ($1).text)
74 [...] Heap Blocks: exact=10
75 [...] > Bitmap Index Scan on index_entrega_productos_ruc (cost=0.00..4.75 rows=100 width=0) (actual time=3.281..3.281 rows=473 loops=1)
76 [...] Index Cond: ((ruc).text = ($1).text)
77 [...] > Index Scan using pk_compra_correlativo on compra c_1 (cost=0.29..0.37 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=486)
78 [...] Index Cond: (correlativo = itemcompra_1.correlativo)
79 [...] Filter: ((EXTRACT(year FROM fechaHora) = ANY ('(2020,2021)::numeric[]')) AND (EXTRACT(month FROM fechaHora) = ANY ('(1,2,3,12,11,10)::numeric[]'))))
80 [...] Rows Removed by Filter: 1
81 Planning Time: 2.214 ms
82 Execution Time: 16.845 ms

```

Figura 40: Query Plan de la consulta 2 de 10k datos con índices

5.3.2.5. Ejecución sin índices para 100k datos

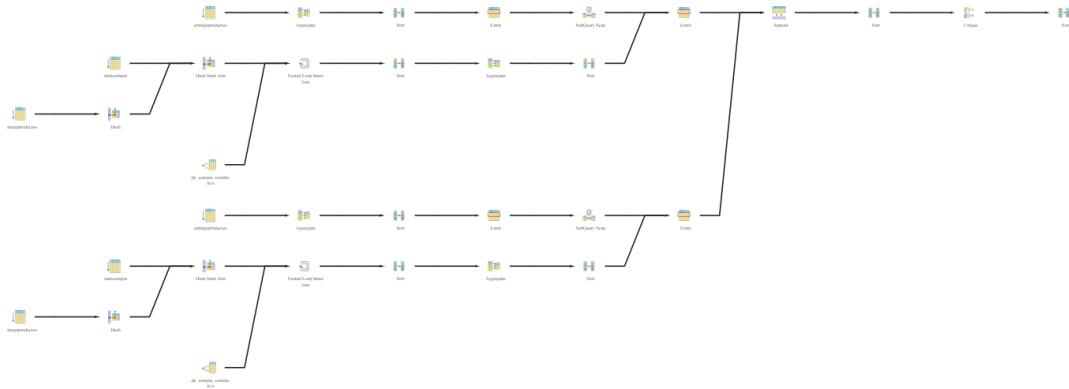


Figura 41: Query Plan Tree de la consulta 2 de 100k datos sin índices

1	Sort (cost=15477.03..15477.04 rows=2 width=86) (actual time=984.622..984.641 rows=20 loops=1)
2	[...] Sort Key: (sum(itemcompra.cantidad))
3	[...] Sort Method: quicksort Memory: 26kB
4	[...] -> Unique (cost=15476.99..15477.00 rows=2 width=86) (actual time=984.571..984.597 rows=20 loops=1)
5	[...] -> Sort (cost=15476.99..15476.99 rows=2 width=86) (actual time=984.569..984.588 rows=20 loops=1)
6	[...] Sort Key: itemcompra.nombre_producto, (sum(itemcompra.cantidad))
7	[...] Sort Method: quicksort Memory: 26kB
8	[...] -> Append (cost=7738.47..15476.98 rows=2 width=86) (actual time=857.893..984.540 rows=20 loops=1)
9	[...] -> Limit (cost=7738.47..7738.47 rows=1 width=17) (actual time=857.891..857.904 rows=10 loops=1)
10	[...] InitPlan 1 (returns \$0)
11	[...] -> Subquery Scan on foo (cost=3127.03..3127.05 rows=1 width=12) (actual time=209.605..209.610 rows=1 loops=1)
12	[...] -> Limit (cost=3127.03..3127.04 rows=1 width=20) (actual time=209.603..209.606 rows=1 loops=1)
13	[...] -> Sort (cost=3127.03..3128.27 rows=494 width=20) (actual time=209.598..209.600 rows=1 loops=1)
14	[...] Sort Key: (count(entregaproductos_2.nombre_producto)) DESC
15	[...] Sort Method: top-N heapsort Memory: 25kB
16	[...] -> HashAggregate (cost=3119.62..3124.57 rows=494 width=20) (actual time=209.060..209.189 rows=497 loops=1)
17	[...] Group Key: entregaproductos_2.ruc
18	[...] Batches: 1 Memory Usage: 105kB
19	[...] -> Seq Scan on entregaproductos entregaproductos_2 (cost=0.00..2939.16 rows=36092 width=21) (actual time=0.025..192.905 rows=41186 loops=1)
20	[...] Filter: (((medida)::text ~~ '%oz'::text) OR (((medida)::text ~~ '%L'::text) OR (((medida)::text ~~ '%ml'::text))
21	[...] Rows Removed by Filter: 58814
22	[...] -> Sort (cost=4611.42..4611.43 rows=1 width=17) (actual time=857.889..857.896 rows=10 loops=1)
23	[...] Sort Key: (sum(itemcompra.cantidad))
24	[...] Sort Method: top-N heapsort Memory: 26kB
25	[...] -> GroupAggregate (cost=4611.39..4611.41 rows=1 width=17) (actual time=857.739..857.834 rows=150 loops=1)
26	[...] Group Key: itemcompra.nombre_producto
27	[...] -> Sort (cost=4611.39..4611.40 rows=1 width=13) (actual time=857.729..857.746 rows=158 loops=1)
28	[...] Sort Key: itemcompra.nombre_producto
29	[...] Sort Method: quicksort Memory: 33kB
30	[...] -> Nested Loop (cost=2442.79..4611.38 rows=1 width=13) (actual time=272.287..857.066 rows=158 loops=1)
31	[...] -> Hash Semi Join (cost=2442.50..4476.28 rows=368 width=17) (actual time=251.452..349.485 rows=1772 loops=1)
32	[...] Hash Cond: ((itemcompra.nombre_producto)::text = (entregaproductos.nombre_producto)::text)
33	[...] -> Seq Scan on itemcompra (cost=0.00..1769.43 rows=96343 width=17) (actual time=0.766..64.337 rows=100000 loops=1)
34	[...] -> Hash (cost=2439.98..2439.98 rows=202 width=9) (actual time=250.591..250.592 rows=1749 loops=1)

35	[...] Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 87kB
36	[...]-> Seq Scan on entregaproductos (cost=0.00..2439.98 rows=202 width=9) (actual time=211.213..249.896 rows=1749 loops=1)
37	[...] Filter: ((ruc)::text = (\$0)::text)
38	[...] Rows Removed by Filter: 98251
39	[...]-> Index Scan using pk_compra_correlativo on compra c (cost=0.29..0.37 rows=1 width=4) (actual time=0.285..0.285 rows=0 loops=1772)
40	[...] Index Cond: (correlativo = itemcompra.correlativo)
41	[...] Filter: ((EXTRACT(year FROM fecha_hora) = ANY ('(2020,2021)::numeric[])) AND (EXTRACT(month FROM fecha_hora) = ANY ('{1,2,3,12,11,10}::numeric[])))
42	[...] Rows Removed by Filter: 1
43	[...]-> Limit (cost=7738.47..7738.47 rows=1 width=17) (actual time=126.620..126.628 rows=10 loops=1)
44	[...] InitPlan 2 (returns \$1)
45	[...]-> Subquery Scan on foo_1 (cost=3127.03..3127.05 rows=1 width=12) (actual time=45.622..45.626 rows=1 loops=1)
46	[...]-> Limit (cost=3127.03..3127.04 rows=1 width=20) (actual time=45.621..45.623 rows=1 loops=1)
47	[...]-> Sort (cost=3127.03..3128.27 rows=494 width=20) (actual time=45.619..45.621 rows=1 loops=1)
48	[...] Sort Key: (count(entregaproductos_3.nombre_producto)) DESC
49	[...] Sort Method: top-N heapsort Memory: 25kB
50	[...]-> HashAggregate (cost=3119.62..3124.57 rows=494 width=20) (actual time=45.376..45.504 rows=497 loops=1)
51	[...] Group Key: entregaproductos_3.ruc
52	[...] Batches: 1 Memory Usage: 105kB
53	[...]-> Seq Scan on entregaproductos_entregaproductos_3 (cost=0.00..2939.16 rows=36092 width=21) (actual time=0.011..30.983 rows=41186 loops=1)
54	[...] Filter: (((medida)::text ~~ '%oz'::text) OR ((medida)::text ~~ '%L'::text) OR ((medida)::text ~~ '%ml'::text))
55	[...] Rows Removed by Filter: 58814
56	[...]-> Sort (cost=4611.42..4611.43 rows=1 width=17) (actual time=126.619..126.621 rows=10 loops=1)
57	[...] Sort Key: (sum(itemcompra_1.cantidad)) DESC
58	[...] Sort Method: top-N heapsort Memory: 26kB
59	[...]-> GroupAggregate (cost=4611.39..4611.41 rows=1 width=17) (actual time=126.454..126.546 rows=150 loops=1)
60	[...] Group Key: itemcompra_1.nombre_producto
61	[...]-> Sort (cost=4611.39..4611.40 rows=1 width=13) (actual time=126.443..126.457 rows=158 loops=1)
62	[...] Sort Key: itemcompra_1.nombre_producto
63	[...] Sort Method: quicksort Memory: 33kB
64	[...]-> Nested Loop (cost=2442.79..4611.38 rows=1 width=13) (actual time=79.720..126.105 rows=158 loops=1)
65	[...]-> Hash Semi Join (cost=2442.50..4476.28 rows=368 width=17) (actual time=79.535..113.023 rows=1772 loops=1)
66	[...] Hash Cond: ((itemcompra_1.nombre_producto)::text = (entregaproductos_1.nombre_producto)::text)
67	[...]-> Seq Scan on itemcompra itemcompra_1 (cost=0.00..1769.43 rows=96343 width=17) (actual time=0.007..11.016 rows=100000 loops=1)
68	[...]-> Hash (cost=2439.98..2439.98 rows=202 width=9) (actual time=79.479..79.479 rows=1749 loops=1)

69	[...] Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 87kB
70	[...]> Seq Scan on entregaproductos entregaproductos_1 (cost=0.00..2439.98 rows=202 width=9) (actual time=46.777..76.847 rows=1749 loops=1)
71	[...] Filter: ((ruc)::text = (\$1)::text)
72	[...] Rows Removed by Filter: 98251
73	[...]> Index Scan using pk_compra_correlativo on compra c_1 (cost=0.29..0.37 rows=1 width=4) (actual time=0.007..0.007 rows=0 loops=1772)
74	[...] Index Cond: (correlativo = itemcompra_1.correlativo)
75	[...] Filter: ((EXTRACT(year FROM fechaHora) = ANY ('(2020,2021)::numeric[]')) AND (EXTRACT(month FROM fechaHora) = ANY ('{1,2,3,12,11,10}::numeric[]')))
76	[...] Rows Removed by Filter: 1
77	Planning Time: 51.422 ms
78	Execution Time: 985.644 ms

Figura 42: Query Plan de la consulta 2 de 100k datos sin índices

5.3.2.6. Ejecución con índices para 100k datos

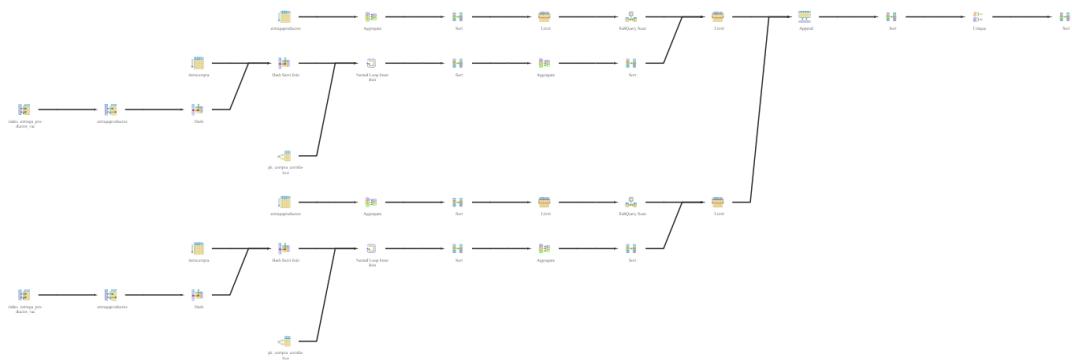


Figura 43: Query Plan Tree de la consulta 2 de 100k datos con índices

1	Sort (cost=11675.91..11675.91 rows=2 width=86) (actual time=238.190..238.212 rows=20 loops=1)
2	[...] Sort Key: (sum(itemcompra.cantidad))
3	[...] Sort Method: quicksort Memory: 26kB
4	[...] -> Unique (cost=11675.86..11675.88 rows=2 width=86) (actual time=238.139..238.166 rows=20 loops=1)
5	[...] -> Sort (cost=11675.86..11675.87 rows=2 width=86) (actual time=238.137..238.158 rows=20 loops=1)
6	[...] Sort Key: itemcompra.nombre_producto, (sum(itemcompra.cantidad))
7	[...] Sort Method: quicksort Memory: 26kB
8	[...] -> Append (cost=5837.91..11675.85 rows=2 width=86) (actual time=154.389..238.134 rows=20 loops=1)
9	[...] -> Limit (cost=5837.91..5837.91 rows=1 width=17) (actual time=154.388..154.401 rows=10 loops=1)
10	[...] InitPlan 1 (returns \$0)
11	[...] -> Subquery Scan on foo (cost=3130.16..3130.18 rows=1 width=12) (actual time=52.348..52.353 rows=1 loops=1)
12	[...] -> Limit (cost=3130.16..3130.17 rows=1 width=20) (actual time=52.347..52.349 rows=1 loops=1)
13	[...] -> Sort (cost=3130.16..3131.40 rows=494 width=20) (actual time=52.345..52.347 rows=1 loops=1)
14	[...] Sort Key: (count(entregaproductos_2.nombre_producto)) DESC
15	[...] Sort Method: top-N heapsort Memory: 25kB
16	[...] -> HashAggregate (cost=3122.76..3127.70 rows=494 width=20) (actual time=52.096..52.234 rows=497 loops=1)
17	[...] Group Key: entregaproductos_2.ruc
18	[...] Batches: 1 Memory Usage: 105kB
19	[...] -> Seq Scan on entregaproductos entregaproductos_2 (cost=0.00..2942.00 rows=36151 width=21) (actual time=0.007..37.053 rows=41186 loops=1)
20	[...] Filter: (((medida)::text ~~ '%oz'::text) OR ((medida)::text ~~ '%L'::text) OR ((medida)::text ~~ '%ml'::text))
21	[...] Rows Removed by Filter: 58814
22	[...] -> Sort (cost=2707.73..2707.73 rows=1 width=17) (actual time=154.386..154.394 rows=10 loops=1)
23	[...] Sort Key: (sum(itemcompra.cantidad))
24	[...] Sort Method: top-N heapsort Memory: 26kB
25	[...] -> GroupAggregate (cost=2707.70..2707.72 rows=1 width=17) (actual time=154.254..154.345 rows=150 loops=1)
26	[...] Group Key: itemcompra.nombre_producto
27	[...] -> Sort (cost=2707.70..2707.70 rows=1 width=13) (actual time=154.244..154.262 rows=158 loops=1)
28	[...] Sort Key: itemcompra.nombre_producto
29	[...] Sort Method: quicksort Memory: 33kB
30	[...] -> Nested Loop (cost=540.71..2707.69 rows=1 width=13) (actual time=55.005..153.987 rows=158 loops=1)
31	[...] -> Hash Semi Join (cost=540.41..2572.94 rows=367 width=17) (actual time=54.329..111.803 rows=1772 loops=1)
32	[...] Hash Cond: ((itemcompra.nombre_producto)::text = (entregaproductos.nombre_producto)::text)
33	[...] -> Seq Scan on itemcompra (cost=0.00..1768.46 rows=96246 width=17) (actual time=0.030..36.512 rows=100000 loops=1)
34	[...] -> Hash (cost=537.89..537.89 rows=202 width=9) (actual time=54.268..54.270 rows=1749 loops=1)

```

35 [...] Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 87kB
36 [...] -> Bitmap Heap Scan on entregaproductos (cost=9.57..537.89 rows=202 width=9) (actual time=52.486..53.017 rows=1749 loops=1)
37 [...] Recheck Cond: ((ruc)::text = ($0)::text)
38 [...] Heap Blocks: exact=48
39 [...] -> Bitmap Index Scan on index_entrega_productos_ruc (cost=0.00..9.52 rows=202 width=0) (actual time=52.471..52.471 rows=1749 loops=1)
40 [...] Index Cond: ((ruc)::text = ($0)::text)
41 [...] -> Index Scan using pk_compra_correlativo on compra c (cost=0.29..0.37 rows=1 width=4) (actual time=0.023..0.023 rows=0 loops=1772)
42 [...] Index Cond: (correlativo = itemcompra.correlativo)
43 [...] Filter: ((EXTRACT(year FROM fechaHora) = ANY ('(2020,2021)::numeric[])) AND (EXTRACT(month FROM fechaHora) = ANY ('1,2,3,12,11,10)::numeric[])))
44 [...] Rows Removed by Filter: 1
45 [...] -> Limit (cost=5837.91..5837.91 rows=1 width=17) (actual time=83.715..83.723 rows=10 loops=1)
46 [...] InitPlan 2 (returns $1)
47 [...] -> Subquery Scan on foo_1 (cost=3130.16..3130.18 rows=1 width=12) (actual time=42.496..42.500 rows=1 loops=1)
48 [...] -> Limit (cost=3130.16..3130.17 rows=1 width=20) (actual time=42.495..42.497 rows=1 loops=1)
49 [...] -> Sort (cost=3130.16..3131.40 rows=494 width=20) (actual time=42.492..42.494 rows=1 loops=1)
50 [...] Sort Key: (count(entregaproductos_3.nombre_producto)) DESC
51 [...] Sort Method: top-N heapsort Memory: 25kB

```

```

52 [...] -> HashAggregate (cost=3122.76..3127.70 rows=494 width=20) (actual time=42.253..42.375 rows=497 loops=1)
53 [...] Group Key: entregaproductos_3.ruc
54 [...] Batches: 1 Memory Usage: 105kB
55 [...] -> Seq Scan on entregaproductos entregaproductos_3 (cost=0.00..2942.00 rows=36151 width=21) (actual time=0.012..30.332 rows=41186 loops=1)
56 [...] Filter: (((medida)::text ~~ '%oz)::text) OR ((medida)::text ~~ '%L)::text) OR ((medida)::text ~~ '%ml)::text)
57 [...] Rows Removed by Filter: 58814
58 [...] -> Sort (cost=2707.73..2707.73 rows=1 width=17) (actual time=83.713..83.716 rows=10 loops=1)
59 [...] Sort Key: (sum(itemcompra_1.cantidad)) DESC
60 [...] Sort Method: top-N heapsort Memory: 26kB
61 [...] -> GroupAggregate (cost=2707.70..2707.72 rows=1 width=17) (actual time=83.566..83.656 rows=150 loops=1)
62 [...] Group Key: itemcompra_1.nombre_producto
63 [...] -> Sort (cost=2707.70..2707.70 rows=1 width=13) (actual time=83.555..83.570 rows=158 loops=1)
64 [...] Sort Key: itemcompra_1.nombre_producto
65 [...] Sort Method: quicksort Memory: 33kB
66 [...] -> Nested Loop (cost=540.71..2707.69 rows=1 width=13) (actual time=43.961..83.310 rows=158 loops=1)
67 [...] -> Hash Semi Join (cost=540.41..2572.94 rows=367 width=17) (actual time=43.781..71.439 rows=1772 loops=1)
68 [...] Hash Cond: ((itemcompra_1.nombre_producto)::text = (entregaproductos_1.nombre_producto)::text)

```

69	[...] > Seq Scan on itemcompra itemcompra_1 (cost=0.00..1768.46 rows=96246 width=17) (actual time=0.010..10.150 rows=100000 loops=1)
70	[...] > Hash (cost=537.89..537.89 rows=202 width=9) (actual time=43.724..43.725 rows=1749 loops=1)
71	[...] Buckets: 2048 (originally 1024). Batches: 1 (originally 1). Memory Usage: 87kB
72	[...] > Bitmap Heap Scan on entregaproductos entregaproductos_1 (cost=9.57..537.89 rows=202 width=9) (actual time=42.644..43.211 rows=1749 loops=1)
73	[...] Recheck Cond: ((ruc)::text = (\$1)::text)
74	[...] Heap Blocks: exact=48
75	[...] > Bitmap Index Scan on index_entrega_productos_ruc (cost=0.00..9.52 rows=202 width=0) (actual time=42.627..42.627 rows=1749 loops=1)
76	[...] Index Cond: ((ruc)::text = (\$1)::text)
77	[...] > Index Scan using pk_c_compra_correlativo on compra c_1 (cost=0.29..0.37 rows=1 width=4) (actual time=0.006..0.006 rows=0 loops=1772)
78	[...] Index Cond: (correlativo = itemcompra_1.correlativo)
79	[...] Filter: ((EXTRACT(year FROM fecha_hora) = ANY ('(2020,2021)::numeric[])) AND (EXTRACT(month FROM fecha_hora) = ANY ('(1,2,3,12,11,10)::numeric[])))
80	[...] Rows Removed by Filter: 1
81	Planning Time: 2.560 ms
82	Execution Time: 238.625 ms

Figura 44: Query Plan de la consulta 2 de 100k datos con índices

5.3.2.7. Ejecución sin índices para 1m datos

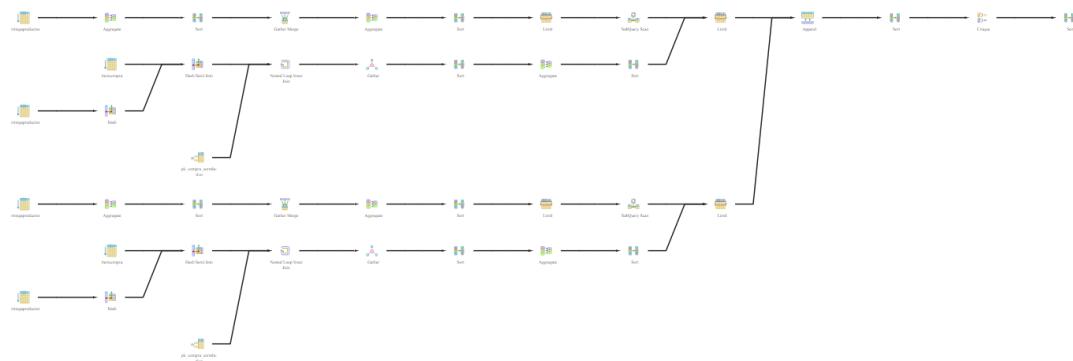


Figura 45: Query Plan Tree de la consulta 2 de 1m datos sin índices

1	Sort (cost=99130.61..99130.61 rows=2 width=86) (actual time=835.553..842.873 rows=20 loops=1)
2	[...] Sort Key: (sum(itemcompra.cantidad))
3	[...] Sort Method: quicksort Memory: 25kB
4	[...]-> Unique (cost=99130.56..99130.58 rows=2 width=86) (actual time=835.525..842.852 rows=20 loops=1)
5	[...]-> Sort (cost=99130.56..99130.57 rows=2 width=86) (actual time=835.524..842.843 rows=20 loops=1)
6	[...] Sort Key: itemcompra.nombre_producto, (sum(itemcompra.cantidad))
7	[...] Sort Method: quicksort Memory: 25kB
8	[...]-> Append (cost=49565.26..99130.55 rows=2 width=86) (actual time=489.152..842.818 rows=20 loops=1)
9	[...]-> Limit (cost=49565.26..49565.26 rows=1 width=15) (actual time=489.151..492.529 rows=10 loops=1)
10	[...] InitPlan 1 (returns \$1)
11	[...]-> Subquery Scan on foo (cost=19569.31..19569.33 rows=1 width=12) (actual time=227.685..230.758 rows=1 loops=1)
12	[...]-> Limit (cost=19569.31..19569.32 rows=1 width=20) (actual time=227.684..230.754 rows=1 loops=1)
13	[...]-> Sort (cost=19569.31..19570.56 rows=500 width=20) (actual time=227.682..230.752 rows=1 loops=1)
14	[...] Sort Key: (count(entregaproductos_2.nombre_producto)) DESC
15	[...] Sort Method: top-N heapsort Memory: 25kB
16	[...]-> Finalize GroupAggregate (cost=19440.14..19566.81 rows=500 width=20) (actual time=226.585..230.622 rows=500 loops=1)
17	[...] Group Key: entregaproductos_2.ruc
18	[...]-> Gather Merge (cost=19440.14..19556.81 rows=1000 width=20) (actual time=226.570..230.177 rows=1500 loops=1)
19	[...] Workers Planned: 2
20	[...] Workers Launched: 2
21	[...]-> Sort (cost=18440.11..18441.36 rows=500 width=20) (actual time=218.305..218.349 rows=500 loops=3)
22	[...] Sort Key: entregaproductos_2.ruc
23	[...] Sort Method: quicksort Memory: 64kB
24	[...] Worker 0: Sort Method: quicksort Memory: 64kB
25	[...] Worker 1: Sort Method: quicksort Memory: 64kB
26	[...]-> Partial HashAggregate (cost=18412.70..18417.70 rows=500 width=20) (actual time=217.162..217.258 rows=500 loops=3)
27	[...] Group Key: entregaproductos_2.ruc
28	[...] Batches: 1 Memory Usage: 105kB
29	[...] Worker 0: Batches: 1 Memory Usage: 105kB
30	[...] Worker 1: Batches: 1 Memory Usage: 105kB
31	[...]-> Parallel Seq Scan on entregaproductos entregaproductos_2 (cost=0.00..17520.19 rows=178503 width=19) (actual time=0.098..166.847 rows=157649 loops=3)
32	[...] Filter: (((medida)::text ~~~ '%oz'::text) OR ((medida)::text ~~ '%L'::text) OR ((medida)::text ~~~ '%ml'::text))
33	[...] Rows Removed by Filter: 175684
34	[...]-> Sort (cost=29995.93..29995.93 rows=1 width=15) (actual time=489.148..489.452 rows=10 loops=1)

35	[...] Sort Key: (sum(itemcompra.cantidad))
36	[...] Sort Method: top-N heapsort Memory: 25kB
37	[...] -> GroupAggregate (cost=29995.90..29995.92 rows=1 width=15) (actual time=489.008..489.397 rows=145 loops=1)
38	[...] Group Key: itemcompra.nombre_producto
39	[...] -> Sort (cost=29995.90..29995.90 rows=1 width=11) (actual time=488.996..489.310 rows=150 loops=1)
40	[...] Sort Key: itemcompra.nombre_producto
41	[...] Sort Method: quicksort Memory: 32kB
42	[...] -> Gather (cost=16447.83..29995.89 rows=1 width=11) (actual time=315.790..488.988 rows=150 loops=1)
43	[...] Workers Planned: 2
44	[...] Params Evaluated: \$1
45	[...] Workers Launched: 2
46	[...] -> Nested Loop (cost=15447.83..28995.79 rows=1 width=11) (actual time=74.572..246.832 rows=50 loops=3)
47	[...] -> Parallel Hash Semi Join (cost=15447.40..28185.86 rows=1625 width=15) (actual time=70.729..215.918 rows=689 loops=3)
48	[...] Hash Cond: ((itemcompra.nombre_producto)::text = (entregaproductos.nombre_producto)::text)
49	[...] -> Parallel Seq Scan on itemcompra (cost=0.00..11627.26 rows=416426 width=15) (actual time=0.141..77.047 rows=333333 loops=3)
50	[...] -> Parallel Hash (cost=15436.99..15436.99 rows=833 width=7) (actual time=69.169..69.170 rows=678 loops=3)
51	[...] Buckets: 2048 Batches: 1 Memory Usage: 144kB
52	[...] -> Parallel Seq Scan on entregaproductos (cost=0.00..15436.99 rows=833 width=7) (actual time=0.152..65.780 rows=678 loops=3)
53	[...] Filter: ((ruc)::text = (\$1)::text)
54	[...] Rows Removed by Filter: 332655
55	[...] -> Index Scan using pk_compra_correlativo on compra c (cost=0.42..0.50 rows=1 width=4) (actual time=0.044..0.044 rows=0 loops=2067)
56	[...] Index Cond: (correlativo = itemcompra.correlativo)
57	[...] Filter: ((EXTRACT(year FROM fecha_hora) = ANY ('(2020,2021)')::numeric[])) AND ((EXTRACT(month FROM fecha_hora) = ANY ('{1,2,3,12,11,10}')::numeric[]))
58	[...] Rows Removed by Filter: 1
59	[...] -> Limit (cost=49565.26..49565.26 rows=1 width=15) (actual time=346.338..350.278 rows=10 loops=1)
60	[...] InitPlan 2 (returns \$3)
61	[...] -> Subquery Scan on foo_1 (cost=19569.31..19569.33 rows=1 width=12) (actual time=170.705..170.767 rows=1 loops=1)
62	[...] -> Limit (cost=19569.31..19569.32 rows=1 width=20) (actual time=170.703..170.764 rows=1 loops=1)
63	[...] -> Sort (cost=19569.31..19570.56 rows=500 width=20) (actual time=170.701..170.760 rows=1 loops=1)
64	[...] Sort Key: (count(entregaproductos_3.nombre_producto)) DESC
65	[...] Sort Method: top-N heapsort Memory: 25kB
66	[...] -> Finalize GroupAggregate (cost=19440.14..19566.81 rows=500 width=20) (actual time=169.782..170.652 rows=500 loops=1)
67	[...] Group Key: entregaproductos_3.ruc
68	[...] -> Gather Merge (cost=19440.14..19556.81 rows=1000 width=20) (actual time=169.772..170.334 rows=1500 loops=1)

89	[...] > Sort (cost=29995.90..29995.90 rows=1 width=11) (actual time=346.102..349.985 rows=150 loops=1)
90	[...] Sort Key: itemcompra_1.nombre_producto
91	[...] Sort Method: quicksort Memory: 32kB
92	[...] > Gather (cost=16447.83..29995.89 rows=1 width=11) (actual time=235.785..349.593 rows=150 loops=1)
93	[...] Workers Planned: 2
94	[...] Params Evaluated: \$3
95	[...] Workers Launched: 2
96	[...] > Nested Loop (cost=15447.83..28995.79 rows=1 width=11) (actual time=59.860..167.859 rows=50 loops=3)
97	[...] > Parallel Hash Semi Join (cost=15447.40..28185.86 rows=1625 width=15) (actual time=58.208..157.904 rows=689 loops=3)
98	[...] Hash Cond: ((itemcompra_1.nombre_producto)::text = (entregaproductos_1.nombre_producto)::text)
99	[...] > Parallel Seq Scan on itemcompra itemcompra_1 (cost=0.00..11627.26 rows=416426 width=15) (actual time=0.063..42.513 rows=333333 loops=3)
100	[...] > Parallel Hash (cost=15436.99..15436.99 rows=833 width=7) (actual time=57.558..57.559 rows=678 loops=3)
101	[...] Buckets: 2048 Batches: 1 Memory Usage: 112kB
102	[...] > Parallel Seq Scan on entregaproductos entregaproductos_1 (cost=0.00..15436.99 rows=833 width=7) (actual time=0.126..57.033 rows=678 loops=3)
103	[...] Filter: ((ruc)::text = (\$3)::text)
104	[...] Rows Removed by Filter: 332655
105	[...] > Index Scan using pk_compra_correlativo on compra c_1 (cost=0.42..0.50 rows=1 width=4) (actual time=0.014..0.014 rows=0 loops=2067)
106	[...] Index Cond: (correlativo = itemcompra_1.correlativo)
107	[...] Filter: ((EXTRACT(year FROM fechaHora) = ANY ('(2020,2021)::numeric[])) AND (EXTRACT(month FROM fechaHora) = ANY ('(1,2,3,12,11,10)::numeric[])))
108	[...] Rows Removed by Filter: 1
109	Planning Time: 4.337 ms
110	Execution Time: 843.500 ms

Figura 46: Query Plan de la consulta 2 de 1m datos sin índices

5.3.2.8. Ejecución con índices para 1m datos



Figura 47: Query Plan Tree de la consulta 2 de 1m datos con índices

1	Sort (cost=78384.74..78384.74 rows=2 width=86) (actual time=750.889..760.893 rows=20 loops=1)
2	[...] Sort Key: (sum(itemcompra.cantidad))
3	[...] Sort Method: quicksort Memory: 25kB
4	[...] -> Unique (cost=78384.69..78384.71 rows=2 width=86) (actual time=750.857..760.867 rows=20 loops=1)
5	[...] -> Sort (cost=78384.69..78384.70 rows=2 width=86) (actual time=750.855..760.858 rows=20 loops=1)
6	[...] Sort Key: itemcompra.nombre_producto, (sum(itemcompra.cantidad))
7	[...] Sort Method: quicksort Memory: 25kB
8	[...] -> Append (cost=39192.32..78384.68 rows=2 width=86) (actual time=444.150..760.804 rows=20 loops=1)
9	[...] -> Limit (cost=39192.32..39192.33 rows=1 width=15) (actual time=444.148..444.568 rows=10 loops=1)
10	[...] InitPlan 1 (returns \$1)
11	[...] -> Subquery Scan on foo (cost=19569.85..19569.87 rows=1 width=12) (actual time=202.539..202.646 rows=1 loops=1)
12	[...] -> Limit (cost=19569.85..19569.86 rows=1 width=20) (actual time=202.537..202.643 rows=1 loops=1)
13	[...] -> Sort (cost=19569.85..19571.10 rows=500 width=20) (actual time=202.536..202.640 rows=1 loops=1)
14	[...] Sort Key: (count(entregaproductos_2.nombre_producto)) DESC
15	[...] Sort Method: top-N heapsort Memory: 25kB
16	[...] -> Finalize GroupAggregate (cost=19440.68..19567.35 rows=500 width=20) (actual time=201.375..202.501 rows=500 loops=1)
17	[...] Group Key: entregaproductos_2.ruc
18	[...] -> Gather Merge (cost=19440.68..19557.35 rows=1000 width=20) (actual time=201.364..202.042 rows=1500 loops=1)
19	[...] Workers Planned: 2
20	[...] Workers Launched: 2
21	[...] -> Sort (cost=18440.66..18441.91 rows=500 width=20) (actual time=189.958..190.001 rows=500 loops=3)
22	[...] Sort Key: entregaproductos_2.ruc
23	[...] Sort Method: quicksort Memory: 64kB
24	[...] Worker 0: Sort Method: quicksort Memory: 64kB
25	[...] Worker 1: Sort Method: quicksort Memory: 64kB
26	[...] -> Partial HashAggregate (cost=18413.24..18418.24 rows=500 width=20) (actual time=189.051..189.146 rows=500 loops=3)
27	[...] Group Key: entregaproductos_2.ruc
28	[...] Batches: 1 Memory Usage: 105kB
29	[...] Worker 0: Batches: 1 Memory Usage: 105kB
30	[...] Worker 1: Batches: 1 Memory Usage: 105kB
31	[...] -> Parallel Seq Scan on entregaproductos entregaproductos_2 (cost=0.00..17520.67 rows=178515 width=19) (actual time=0.064..133.538 rows=157649 loops=3)
32	[...] Filter: (((medida).text ~~ "%oz":text) OR ((medida).text ~~ "%L":text) OR ((medida).text ~~ "%ml":text))
33	[...] Rows Removed by Filter: 175684
34	[...] -> Sort (cost=19622.45..19622.46 rows=1 width=15) (actual time=444.145..444.457 rows=10 loops=1)

35	[...] Sort Key: (sum(itemcompra.cantidad))
36	[...] Sort Method: top-N heapsort Memory: 25kB
37	[...]-> GroupAggregate (cost=19622.42..19622.44 rows=1 width=15) (actual time=444.039..444.411 rows=145 loops=1)
38	[...] Group Key: itemcompra.nombre_producto
39	[...]-> Sort (cost=19622.42..19622.43 rows=1 width=11) (actual time=444.023..444.340 rows=150 loops=1)
40	[...] Sort Key: itemcompra.nombre_producto
41	[...] Sort Method: quicksort Memory: 32kB
42	[...]-> Gather (cost=6074.44..19622.41 rows=1 width=11) (actual time=244.445..444.041 rows=150 loops=1)
43	[...] Workers Planned: 2
44	[...] Params Evaluated: \$1
45	[...] Workers Launched: 2
46	[...]-> Nested Loop (cost=5074.44..18622.31 rows=1 width=11) (actual time=25.798..234.174 rows=50 loops=3)
47	[...]-> Parallel Hash Semi Join (cost=5074.01..17812.38 rows=1625 width=15) (actual time=16.633..180.850 rows=689 loops=3)
48	[...] Hash Cond: ((itemcompra.nombre_producto)::text = (entregaproductos.nombre_producto)::text)
49	[...]-> Parallel Seq Scan on itemcompra (cost=0.00..11627.19 rows=416419 width=15) (actual time=0.106..85.736 rows=333333 loops=3)
50	[...]-> Parallel Hash (cost=5059.31..5059.31 rows=1176 width=7) (actual time=14.575..14.577 rows=678 loops=3)
51	[...] Buckets: 2048 Batches: 1 Memory Usage: 144kB
52	[...]-> Parallel Bitmap Heap Scan on entregaproductos (cost=63.50..5059.31 rows=1176 width=7) (actual time=0.283..12.227 rows=678 loops=3)
53	[...] Recheck Cond: ((ruc)::text = (\$1)::text)
54	[...] Heap Blocks: exact=1114
55	[...]-> Bitmap Index Scan on index_entrega_productos_ruc (cost=0.00..63.00 rows=2000 width=0) (actual time=0.413..0.413 rows=2034 loops=1)
56	[...] Index Cond: ((ruc)::text = (\$1)::text)
57	[...]-> Index Scan using pk_compra_correlativo on compra c (cost=0.42..0.50 rows=1 width=4) (actual time=0.076..0.076 rows=0 loops=2067)
58	[...] Index Cond: (correlativo = itemcompra.correlativo)
59	[...] Filter: ((EXTRACT(year FROM fecha_hora) = ANY ('{2020,2021}':numeric[])) AND (EXTRACT(month FROM fecha_hora) = ANY ('{1,2,3,12,11,10}':numeric[])))
60	[...] Rows Removed by Filter: 1
61	[...]-> Limit (cost=39192.32..39192.33 rows=1 width=15) (actual time=306.644..316.226 rows=10 loops=1)
62	[...] InitPlan 2 (returns \$3)
63	[...]-> Subquery Scan on foo_1 (cost=19569.85..19569.87 rows=1 width=12) (actual time=190.171..190.247 rows=1 loops=1)
64	[...]-> Limit (cost=19569.85..19569.86 rows=1 width=20) (actual time=190.168..190.242 rows=1 loops=1)
65	[...]-> Sort (cost=19569.85..19571.10 rows=500 width=20) (actual time=190.138..190.210 rows=1 loops=1)
66	[...] Sort Key: (count(entregaproductos_3.nombre_producto)) DESC
67	[...] Sort Method: top-N heapsort Memory: 25kB
68	[...]-> Finalize GroupAggregate (cost=19440.68..19567.35 rows=500 width=20) (actual time=189.039..190.093 rows=500 loops=1)

69	[...] Group Key: entregaproductos_3.ruc
70	[...]> Gather Merge (cost=19440.68..19557.35 rows=1000 width=20) (actual time=189.026..189.701 rows=1500 loops=1)
71	[...] Workers Planned: 2
72	[...] Workers Launched: 2
73	[...]> Sort (cost=18440.66..18441.91 rows=500 width=20) (actual time=176.252..176.300 rows=500 loops=3)
74	[...] Sort Key: entregaproductos_3.ruc
75	[...] Sort Method: quicksort Memory: 64kB
76	[...] Worker 0: Sort Method: quicksort Memory: 64kB
77	[...] Worker 1: Sort Method: quicksort Memory: 64kB
78	[...]> Partial HashAggregate (cost=18413.24..18418.24 rows=500 width=20) (actual time=175.551..175.639 rows=500 loops=3)
79	[...] Group Key: entregaproductos_3.ruc
80	[...] Batches: 1 Memory Usage: 105kB
81	[...] Worker 0: Batches: 1 Memory Usage: 105kB
82	[...] Worker 1: Batches: 1 Memory Usage: 105kB
83	[...]> Parallel Seq Scan on entregaproductos entregaproductos_3 (cost=0.00..17520.67 rows=178515 width=19) (actual time=0.130..119.497 rows=157649 loops=3)
84	[...] Filter: (((medida):text ~~ '%oz':text) OR ((medida):text ~~ '%L':text) OR ((medida):text ~~ '%ml':text))
85	[...] Rows Removed by Filter: 175684
86	[...]> Sort (cost=19622.45..19622.46 rows=1 width=15) (actual time=306.642..316.147 rows=10 loops=1)
87	[...] Sort Key: (sum(itemcompra_1.cantidad)) DESC
88	[...] Sort Method: top-N heapsort Memory: 25kB
89	[...]> GroupAggregate (cost=19622.42..19622.44 rows=1 width=15) (actual time=306.492..316.085 rows=145 loops=1)
90	[...] Group Key: itemcompra_1.nombre_producto
91	[...]> Sort (cost=19622.42..19622.43 rows=1 width=11) (actual time=306.481..315.996 rows=150 loops=1)
92	[...] Sort Key: itemcompra_1.nombre_producto
93	[...] Sort Method: quicksort Memory: 32kB
94	[...]> Gather (cost=6074.44..19622.41 rows=1 width=11) (actual time=203.336..315.290 rows=150 loops=1)
95	[...] Workers Planned: 2
96	[...] Params Evaluated: \$3
97	[...] Workers Launched: 2
98	[...]> Nested Loop (cost=5074.44..18622.31 rows=1 width=11) (actual time=4.465..107.586 rows=50 loops=3)
99	[...]> Parallel Hash Semi Join (cost=5074.01..17812.38 rows=1625 width=15) (actual time=2.228..97.447 rows=689 loops=3)
100	[...] Hash Cond: ((itemcompra_1.nombre_producto):text = (entregaproductos_1.nombre_producto):text)
101	[...]> Parallel Seq Scan on itemcompra itemcompra_1 (cost=0.00..11627.19 rows=416419 width=15) (actual time=0.046..40.182 rows=333333 loops=3)
102	[...]> Parallel Hash (cost=5059.31..5059.31 rows=1176 width=7) (actual time=1.916..1.917 rows=678 loops=3)
103	[...] Buckets: 2048 Batches: 1 Memory Usage: 112kB
104	[...]> Parallel Bitmap Heap Scan on entregaproductos entregaproductos_1 (cost=63.50..5059.31 rows=1176 width=7) (actual time=0.713..5.243 rows=2034 loops=1)
105	[...] Recheck Cond: ((ruc):text = (\$3):text)
106	[...] Heap Blocks: exact=1853
107	[...]> Bitmap Index Scan on index_entrega_productos_ruc (cost=0.00..63.00 rows=2000 width=0) (actual time=0.381..0.381 rows=2034 loops=1)
108	[...] Index Cond: ((ruc):text = (\$3):text)
109	[...]> Index Scan using pk_compra_correlativo on compra c_1 (cost=0.42..0.50 rows=1 width=4) (actual time=0.014..0.014 rows=0 loops=2067)
110	[...] Index Cond: (correlativo = itemcompra_1.correlativo)
111	[...] Filter: ((EXTRACT(year FROM fecha_hora) = ANY ('2020,2021')::numeric[])) AND ((EXTRACT(month FROM fecha_hora) = ANY ('1,2,3,12,11,10')::numeric[])))
112	[...] Rows Removed by Filter: 1
113	Planning Time: 3.106 ms
114	Execution Time: 762.338 ms

Figura 48: Query Plan de la consulta 2 de 1m datos con índices

5.3.2.9. Explicación de la consulta

Sin índices: En los planes de consulta para las tablas 1k, 10k, 100k y 1m sin índices podemos hallar que se utilizan las primary key que poseen índice btree por defecto si es que los atributos a analizar lo poseen. En la consulta sin índices, pgadmin4 trata de hacer un hash con un atributo de cierta tabla, esto hace que la consulta sea lenta.

Con índices: Para los planes de consulta con índices, se creó exclusivamente un índice del tipo hash: index_entrega_productos_ruc. Este índice ayuda a optimizar el renderimiento de la consulta, ya que se realiza de manera más rápida un hash semi join y es lo que pgadmin4 necesitaba inicialmente.

5.3.3. Planes de índices para Consulta 3

```
1 CREATE INDEX idx_delivery ON delivery USING hash(correlativo);
2 CREATE INDEX idx_itemcompra ON itemcompra USING hash(correlativo);
3 CREATE INDEX idx_persona ON persona USING hash(dni);
```

5.3.3.1. Ejecución sin índices para 1k datos

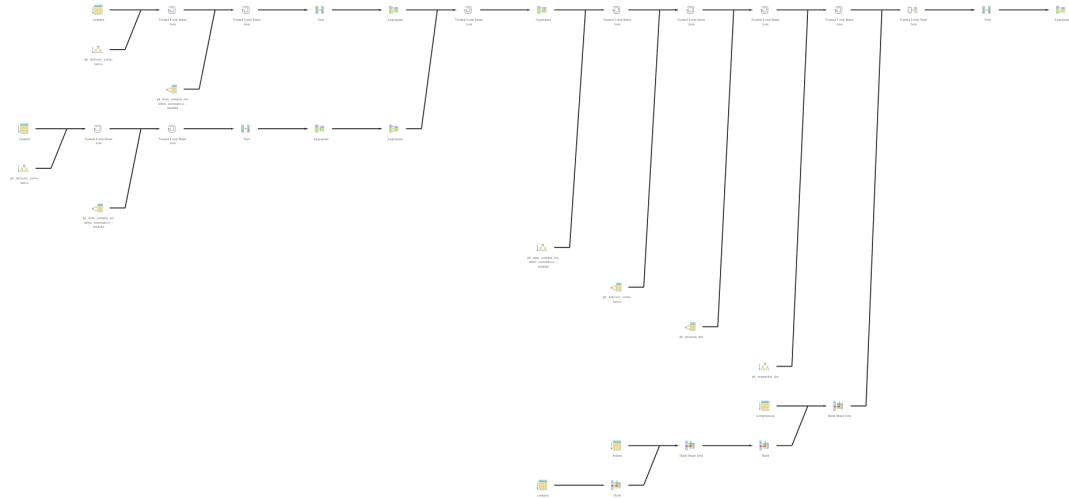


Figura 49: Query Plan Tree de la consulta 3 de 1k datos sin índices

1	GroupAggregate (cost=153.79..153.82 rows=1 width=31) (actual time=5.826..5.852 rows=3 loops=1)
2	[...] Group Key: r.dni, p2.nombre, i.nombre_producto
3	[...] > Sort (cost=153.79..153.80 rows=1 width=23) (actual time=5.812..5.836 rows=3 loops=1)
4	[...] Sort Key: r.dni, p2.nombre, i.nombre_producto
5	[...] Sort Method: quicksort Memory: 25kB
6	[...] > Nested Loop Semi Join (cost=130.87..153.78 rows=1 width=23) (actual time=4.820..5.581 rows=3 loops=1)
7	[...] Join Filter: ((d.dni)::text = (b.dni)::text)
8	[...] Rows Removed by Join Filter: 828
9	[...] > Nested Loop (cost=91.63..96.24 rows=1 width=32) (actual time=3.054..3.487 rows=3 loops=1)
10	[...] Join Filter: ((d.rdn1)::text = (r.dni)::text)
11	[...] > Nested Loop (cost=91.49..96.06 rows=1 width=41) (actual time=3.035..3.450 rows=3 loops=1)
12	[...] > Nested Loop (cost=91.22..95.65 rows=1 width=26) (actual time=2.995..3.347 rows=3 loops=1)
13	[...] > Nested Loop (cost=90.94..95.01 rows=2 width=12) (actual time=2.962..2.995 rows=5 loops=1)
14	[...] > HashAggregate (cost=90.67..90.68 rows=1 width=8) (actual time=2.769..2.791 rows=1 loops=1)
15	[...] Group Key: (i_1.nombre_producto)::text
16	[...] Batches: 1 Memory Usage: 24kB
17	[...] > Nested Loop (cost=90.60..90.67 rows=1 width=8) (actual time=2.760..2.778 rows=1 loops=1)
18	[...] Join Filter: ((sum(i_1.cantidad)) = (max((sum(i_2.cantidad)))))
19	[...] Rows Removed by Join Filter: 8
20	[...] > GroupAggregate (cost=45.29..45.31 rows=1 width=16) (actual time=1.290..1.306 rows=9 loops=1)
21	[...] Group Key: i_1.nombre_producto
22	[...] > Sort (cost=45.29..45.29 rows=1 width=12) (actual time=1.279..1.288 rows=9 loops=1)
23	[...] Sort Key: i_1.nombre_producto
24	[...] Sort Method: quicksort Memory: 25kB
25	[...] > Nested Loop (cost=0.55..45.28 rows=1 width=12) (actual time=0.133..1.171 rows=9 loops=1)
26	[...] > Nested Loop (cost=0.27..37.31 rows=1 width=8) (actual time=0.111..0.583 rows=13 loops=1)
27	[...] > Seq Scan on compra c_1 (cost=0.00..33.00 rows=1 width=4) (actual time=0.053..0.479 rows=37 loops=1)
28	[...] Filter: ((EXTRACT(year FROM fechaHora) = '2021'::numeric) AND (EXTRACT(month FROM fechaHora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
29	[...] Rows Removed by Filter: 963
30	[...] > Index Only Scan using pk_delivery_correlativo on delivery d_1 (cost=0.27..4.29 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=37)
31	[...] Index Cond: (correlativo = c_1.correlativo)
32	[...] Heap Fetches: 0
33	[...] > Index Scan using pk_item_compra_nombre_correlativo_medida on itemcompra l_1 (cost=0.28..7.95 rows=2 width=16) (actual time=0.032..0.044 rows=1 loops=13)
34	[...] Index Cond: (correlativo = d_1.correlativo)
35	[...] > Aggregate (cost=45.32..45.33 rows=1 width=8) (actual time=0.161..0.162 rows=1 loops=9)
36	[...] > GroupAggregate (cost=45.29..45.31 rows=1 width=16) (actual time=0.148..0.160 rows=9 loops=9)
37	[...] Group Key: i_2.nombre_producto
38	[...] > Sort (cost=45.29..45.29 rows=1 width=12) (actual time=0.146..0.147 rows=9 loops=9)
39	[...] Sort Key: i_2.nombre_producto
40	[...] Sort Method: quicksort Memory: 25kB

41	[...] -> Nested Loop (cost=0.55..45.28 rows=1 width=12) (actual time=0.069..1.124 rows=9 loops=1)
42	[...] -> Nested Loop (cost=0.27..37.31 rows=1 width=8) (actual time=0.055..0.510 rows=13 loops=1)
43	[...] -> Seq Scan on compra c_2 (cost=0.00..33.00 rows=1 width=4) (actual time=0.021..0.429 rows=37 loops=1)
44	[...] Filter: ((EXTRACT(year FROM fechaHora) = '2021'::numeric) AND (EXTRACT(month FROM fechaHora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
45	[...] Rows Removed by Filter: 963
46	[...] -> Index Only Scan using pk_delivery_correlativo on delivery d_2 (cost=0.27..4.29 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=37)
47	[...] Index Cond: (correlativo = c_2.correlativo)
48	[...] Heap Fetches: 0
49	[...] -> Index Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i_2 (cost=0.28..7.95 rows=2 width=16) (actual time=0.037..0.046 rows=1 loops=13)
50	[...] Index Cond: (correlativo = d_2.correlativo)
51	[...] -> Index Only Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i (cost=0.28..4.31 rows=2 width=12) (actual time=0.188..0.196 rows=5 loops=1)
52	[...] Index Cond: (nombre_producto = (i_1.nombre_producto)::text)
53	[...] Heap Fetches: 0
54	[...] -> Index Scan using pk_delivery_correlativo on delivery d (cost=0.27..0.32 rows=1 width=22) (actual time=0.067..0.067 rows=1 loops=5)
55	[...] Index Cond: (correlativo = i.correlativo)
56	[...] -> Index Scan using pk_persona_dni on persona p2 (cost=0.28..0.41 rows=1 width=15) (actual time=0.030..0.030 rows=1 loops=3)
57	[...] Index Cond: ((dni)::text = (d.dni)::text)
58	[...] -> Index Only Scan using pk_repartidor_dni on repartidor r (cost=0.14..0.17 rows=1 width=9) (actual time=0.009..0.009 rows=1 loops=3)
59	[...] Index Cond: (dni = (p2.dni)::text)
60	[...] Heap Fetches: 0
61	[...] -> Hash Join (cost=39.24..55.29 rows=180 width=9) (actual time=0.512..0.663 rows=277 loops=3)
62	[...] Hash Cond: (cl.correlativo = b.correlativo)
63	[...] -> Seq Scan on compralocal cl (cost=0.00..12.00 rows=600 width=4) (actual time=0.005..0.039 rows=351 loops=3)
64	[...] -> Hash (cost=36.99..36.99 rows=180 width=17) (actual time=1.493..1.495 rows=478 loops=1)
65	[...] Buckets: 1024 Batches: 1 Memory Usage: 33kB
66	[...] -> Hash Join (cost=27.16..36.99 rows=180 width=17) (actual time=1.166..1.385 rows=478 loops=1)
67	[...] Hash Cond: (b.correlativo = c.correlativo)
68	[...] -> Seq Scan on boleta b (cost=0.00..8.40 rows=540 width=13) (actual time=0.024..0.072 rows=540 loops=1)
69	[...] -> Hash (cost=23.00..23.00 rows=333 width=4) (actual time=0.534..0.535 rows=892 loops=1)
70	[...] Buckets: 1024 Batches: 1 Memory Usage: 40kB
71	[...] -> Seq Scan on compra c (cost=0.00..23.00 rows=333 width=4) (actual time=0.021..0.360 rows=892 loops=1)
72	[...] Filter: (EXTRACT(year FROM fechaHora) < '2021'::numeric)
73	[...] Rows Removed by Filter: 108
74	Planning Time: 9.124 ms
75	Execution Time: 6.579 ms

Figura 50: Query Plan de la consulta 3 de 1k datos sin índices

5.3.3.2. Ejecución con índices para 1k datos

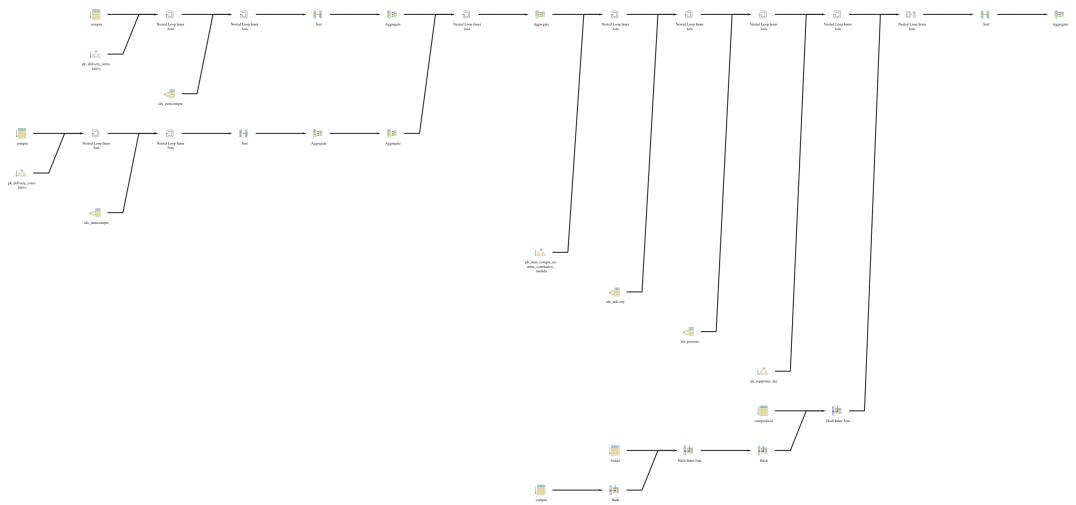


Figura 51: Query Plan Tree de la consulta 3 de 1k datos con índices

```

1  GroupAggregate (cost=137.43..137.46 rows=1 width=31) (actual time=2.948..2.962 rows=3 loops=1)
2  [...] Group Key: r.dni, p2.nombre, i.nombre_producto
3  [...]-> Sort (cost=137.43..137.44 rows=1 width=23) (actual time=2.940..2.952 rows=3 loops=1)
4  [...] Sort Key: r.dni, p2.nombre, i.nombre_producto
5  [...] Sort Method: quicksort Memory: 25kB
6  [...]-> Nested Loop Semi Join (cost=114.78..137.42 rows=1 width=23) (actual time=2.550..2.925 rows=3 loops=1)
7  [...] Join Filter: ((c.dni)::text = (b.dni)::text)
8  [...] Rows Removed by Join Filter: 828
9  [...]-> Nested Loop (cost=75.54..79.88 rows=1 width=32) (actual time=1.189..1.261 rows=3 loops=1)
10 [...] Join Filter: ((d.rdn1)::text = (r.dni)::text)
11 [...]-> Nested Loop (cost=75.40..79.70 rows=1 width=41) (actual time=1.173..1.228 rows=3 loops=1)
12 [...]-> Nested Loop (cost=75.40..79.57 rows=1 width=26) (actual time=1.151..1.189 rows=3 loops=1)
13 [...]-> Nested Loop (cost=75.40..79.47 rows=2 width=12) (actual time=1.139..1.157 rows=5 loops=1)
14 [...]-> HashAggregate (cost=75.13..75.14 rows=1 width=8) (actual time=1.119..1.128 rows=1 loops=1)
15 [...] Group Key: (i_1.nombre_producto)::text
16 [...] Batches: 1 Memory Usage: 24kB
17 [...]-> Nested Loop (cost=75.06..75.13 rows=1 width=8) (actual time=1.113..1.121 rows=1 loops=1)
18 [...] Join Filter: ((sum(i_1.cantidad)) = (max((sum(i_2.cantidad))))) )
19 [...] Rows Removed by Join Filter: 8
20 [...]-> GroupAggregate (cost=37.52..37.54 rows=1 width=16) (actual time=0.563..0.573 rows=9 loops=1)
21 [...] Group Key: i_1.nombre_producto
22 [...]-> Sort (cost=37.52..37.52 rows=1 width=12) (actual time=0.557..0.562 rows=9 loops=1)
23 [...] Sort Key: i_1.nombre_producto
24 [...] Sort Method: quicksort Memory: 25kB
25 [...]-> Nested Loop (cost=0.27..37.51 rows=1 width=12) (actual time=0.103..0.533 rows=9 loops=1)
26 [...]-> Nested Loop (cost=0.27..37.31 rows=1 width=8) (actual time=0.079..0.463 rows=13 loops=1)
27 [...]-> Seq Scan on compra c_1 (cost=0.00..33.00 rows=1 width=4) (actual time=0.037..0.380 rows=37 loops=1)
28 [...] Filter: ((EXTRACT(year FROM fechaHora) = '2021'::numeric) AND (EXTRACT(month FROM fechaHora) = ANY ('(1,2,3,4,5,6)'::numeric[])))
29 [...] Rows Removed by Filter: 963
30 [...]-> Index Only Scan using pk_delivery_correlativo on delivery d_1 (cost=0.27..4.29 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=37)
31 [...] Index Cond: (correlativo = c_1.correlativo)
32 [...] Heap Fetches: 0
33 [...]-> Index Scan using idx_itemcompra on itemcompra i_1 (cost=0.00..0.17 rows=2 width=16) (actual time=0.004..0.005 rows=1 loops=13)
34 [...] Index Cond: (correlativo = d_1.correlativo)
35 [...]-> Aggregate (cost=37.55..37.56 rows=1 width=8) (actual time=0.060..0.060 rows=1 loops=9)
36 [...]-> GroupAggregate (cost=37.52..37.54 rows=1 width=16) (actual time=0.054..0.059 rows=9 loops=9)
37 [...] Group Key: i_2.nombre_producto
38 [...]-> Sort (cost=37.52..37.52 rows=1 width=12) (actual time=0.047..0.048 rows=9 loops=9)
39 [...] Sort Key: i_2.nombre_producto
40 [...] Sort Method: quicksort Memory: 25kB

```

41	[...] > Nested Loop (cost=0.27..37.51 rows=1 width=12) (actual time=0.042..0.415 rows=9 loops=1)
42	[...] > Nested Loop (cost=0.27..37.31 rows=1 width=8) (actual time=0.038..0.385 rows=13 loops=1)
43	[...] > Seq Scan on compra c_2 (cost=0.00..33.00 rows=1 width=4) (actual time=0.008..0.316 rows=37 loops=1)
44	[...] Filter: ((EXTRACT(year FROM fecha_hora) = '2021'::numeric) AND (EXTRACT(month FROM fecha_hora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
45	[...] Rows Removed by Filter: 963
46	[...] > Index Only Scan using pk_delivery_correlativo on delivery d_2 (cost=0.27..4.29 rows=1 width=4) (actual time=0.001..0.001 rows=0 loops=37)
47	[...] Index Cond: (correlativo = c_2.correlativo)
48	[...] Heap Fetches: 0
49	[...] > Index Scan using idx_itemcompra on itemcompra i_2 (cost=0.00..0.17 rows=2 width=16) (actual time=0.001..0.002 rows=1 loops=13)
50	[...] Index Cond: (correlativo = d_2.correlativo)
51	[...] > Index Only Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i (cost=0.28..4.31 rows=2 width=12) (actual time=0.017..0.023 rows=5 loops=1)
52	[...] Index Cond: (nombre_producto = (i_1.nombre_producto)::text)
53	[...] Heap Fetches: 0
54	[...] > Index Scan using idx_delivery on delivery d (cost=0.00..0.05 rows=1 width=22) (actual time=0.004..0.004 rows=1 loops=5)
55	[...] Index Cond: (correlativo = i.correlativo)
56	[...] > Index Scan using idx_persona on persona p2 (cost=0.00..0.14 rows=1 width=15) (actual time=0.010..0.010 rows=1 loops=3)
57	[...] Index Cond: ((dni)::text = (d.rdn)::text)
58	[...] > Index Only Scan using pk_repartidor_dni on repartidor r (cost=0.14..0.17 rows=1 width=9) (actual time=0.008..0.008 rows=1 loops=3)
59	[...] Index Cond: (dni = (p2.dni)::text)
60	[...] Heap Fetches: 0
61	[...] > Hash Join (cost=39.24..55.29 rows=180 width=9) (actual time=0.386..0.521 rows=277 loops=3)
62	[...] Hash Cond: (cl.correlativo = b.correlativo)
63	[...] > Seq Scan on compralocal cl (cost=0.00..12.00 rows=600 width=4) (actual time=0.004..0.045 rows=351 loops=3)
64	[...] > Hash (cost=36.99..36.99 rows=180 width=17) (actual time=1.130..1.131 rows=478 loops=1)
65	[...] Buckets: 1024 Batches: 1 Memory Usage: 33kB
66	[...] > Hash Join (cost=27.16..36.99 rows=180 width=17) (actual time=0.717..0.953 rows=478 loops=1)
67	[...] Hash Cond: (b.correlativo = c.correlativo)
68	[...] > Seq Scan on boleta b (cost=0.00..8.40 rows=540 width=13) (actual time=0.007..0.064 rows=540 loops=1)
69	[...] > Hash (cost=23.00..23.00 rows=333 width=4) (actual time=0.689..0.690 rows=892 loops=1)
70	[...] Buckets: 1024 Batches: 1 Memory Usage: 40kB
71	[...] > Seq Scan on compra c (cost=0.00..23.00 rows=333 width=4) (actual time=0.009..0.534 rows=892 loops=1)
72	[...] Filter: (EXTRACT(year FROM fecha_hora) < '2021'::numeric)
73	[...] Rows Removed by Filter: 108
74	Planning Time: 3.590 ms
75	Execution Time: 3.258 ms

Figura 52: Query Plan de la consulta 3 de 1k datos con índices

5.3.3.3. Ejecución sin índices para 10k datos

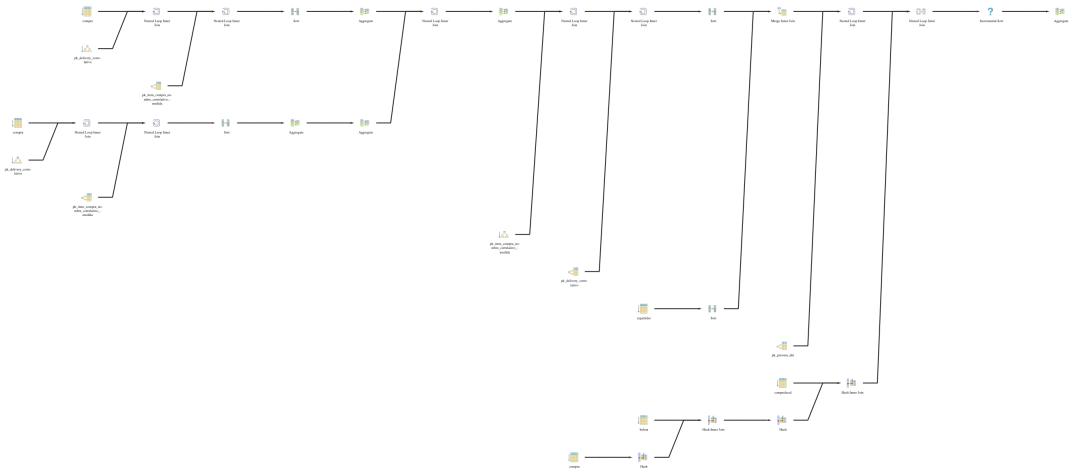


Figura 53: Query Plan Tree de la consulta 3 de 10k datos sin índices

1	GroupAggregate (cost=1285.44..1375.77 rows=1 width=30) (actual time=243.745..243.776 rows=9 loops=1)
2	[...] Group Key: r.dni, p2.nombre, i.nombre_producto
3	[...] > Incremental Sort (cost=1285.44..1375.74 rows=2 width=22) (actual time=243.730..243.755 rows=9 loops=1)
4	[...] Sort Key: r.dni, p2.nombre, i.nombre_producto
5	[...] Presorted Key: r.dni
6	[...] Full-sort Groups: 1 Sort Method: quicksort Average Memory: 25kB Peak Memory: 25kB
7	[...] > Nested Loop Semi Join (cost=1195.21..1375.65 rows=1 width=22) (actual time=196.120..243.038 rows=9 loops=1)
8	[...] Join Filter: ((d.cdni)::text = (b.dni)::text)
9	[...] Rows Removed by Join Filter: 73178
10	[...] > Nested Loop (cost=810.62..811.05 rows=1 width=31) (actual time=182.967..183.870 rows=21 loops=1)
11	[...] > Merge Join (cost=810.33..810.65 rows=1 width=34) (actual time=182.906..183.169 rows=21 loops=1)
12	[...] Merge Cond: ((d.rdn1)::text = (r.dni)::text)
13	[...] > Sort (cost=806.96..806.97 rows=1 width=25) (actual time=182.819..182.875 rows=21 loops=1)
14	[...] Sort Key: d.rdn1
15	[...] Sort Method: quicksort Memory: 26kB
16	[...] > Nested Loop (cost=802.53..806.95 rows=1 width=25) (actual time=182.355..182.681 rows=21 loops=1)
17	[...] > Nested Loop (cost=802.25..806.32 rows=2 width=11) (actual time=182.320..182.458 rows=37 loops=1)
18	[...] > HashAggregate (cost=801.97..801.98 rows=1 width=7) (actual time=181.948..181.969 rows=15 loops=1)
19	[...] Group Key: (l_1.nombre_producto)::text
20	[...] Batches: 1 Memory Usage: 24kB
21	[...] > Nested Loop (cost=801.90..801.96 rows=1 width=7) (actual time=162.493..181.921 rows=15 loops=1)
22	[...] Join Filter: ((sum(l_1.cantidad)) = (max((sum(l_2.cantidad)))))
23	[...] Rows Removed by Join Filter: 149
24	[...] > GroupAggregate (cost=400.93..400.95 rows=1 width=15) (actual time=100.141..100.284 rows=164 loops=1)
25	[...] Group Key: l_1.nombre_producto
26	[...] > Sort (cost=400.93..400.94 rows=1 width=11) (actual time=100.127..100.170 rows=164 loops=1)
27	[...] Sort Key: l_1.nombre_producto
28	[...] Sort Method: quicksort Memory: 33kB
29	[...] > Nested Loop (cost=0.57..400.92 rows=1 width=11) (actual time=0.189..99.848 rows=164 loops=1)
30	[...] > Nested Loop (cost=0.28..326.00 rows=1 width=8) (actual time=0.085..4.887 rows=148 loops=1)
31	[...] > Seq Scan on compra c_1 (cost=0.00..321.70 rows=1 width=4) (actual time=0.057..3.489 rows=411 loops=1)
32	[...] Filter: ((EXTRACT(year FROM fecha_hora) = '2021'::numeric) AND (EXTRACT(month FROM fecha_hora) = ANY ('(1,2,3,4,5,6)'::numeric[])))
33	[...] Rows Removed by Filter: 9589
34	[...] > Index Only Scan using pk_delivery_correlativo on delivery d_1 (cost=0.28..4.30 rows=1 width=4) (actual time=0.003..0.003 rows=0 loops=411)
35	[...] Index Cond: (correlativo = c_1.correlativo)
36	[...] Heap Fetches: 0
37	[...] > Index Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i_1 (cost=0.29..74.91 rows=2 width=15) (actual time=0.362..0.640 rows=1 loops=148)
38	[...] Index Cond: (correlativo = d_1.correlativo)
39	[...] > Aggregate (cost=400.97..400.98 rows=1 width=8) (actual time=0.497..0.497 rows=1 loops=164)
40	[...] > GroupAggregate (cost=400.93..400.95 rows=1 width=15) (actual time=0.371..0.483 rows=164 loops=164)

41	[...] Group Key: i_2.nombre_producto
42	[...] -> Sort (cost=400.93..400.94 rows=1 width=11) (actual time=0.371..0.383 rows=164 loops=164)
43	[...] Sort Key: i_2.nombre_producto
44	[...] Sort Method: quicksort Memory: 33kB
45	[...] -> Nested Loop (cost=0.57..400.92 rows=1 width=11) (actual time=0.140..57.655 rows=164 loops=1)
46	[...] -> Nested Loop (cost=0.28..326.00 rows=1 width=8) (actual time=0.069..4.673 rows=148 loops=1)
47	[...] -> Seq Scan on compra c_2 (cost=0.00..321.70 rows=1 width=4) (actual time=0.047..3.609 rows=411 loops=1)
48	[...] Filter: ((EXTRACT(year FROM fecha_hora) = '2021'::numeric) AND (EXTRACT(month FROM fecha_hora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
49	[...] Rows Removed by Filter: 9589
50	[...] -> Index Only Scan using pk_delivery_correlativo on delivery d_2 (cost=0.28..4.30 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=411)
51	[...] Index Cond: (correlativo = c_2.correlativo)
52	[...] Heap Fetches: 0
53	[...] -> Index Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i_2 (cost=0.29..74.91 rows=2 width=15) (actual time=0.237..0.356 rows=1 loops=148)
54	[...] Index Cond: (correlativo = d_2.correlativo)
55	[...] -> Index Only Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i (cost=0.29..4.32 rows=2 width=11) (actual time=0.030..0.031 rows=2 loops=15)
56	[...] Index Cond: (nombre_producto = (i_1.nombre_producto)::text)
57	[...] Heap Fetches: 0
58	[...] -> Index Scan using pk_delivery_correlativo on delivery d (cost=0.28..0.32 rows=1 width=22) (actual time=0.005..0.005 rows=1 loops=37)
59	[...] Index Cond: (correlativo = i.correlativo)
60	[...] -> Sort (cost=3.37..3.52 rows=60 width=9) (actual time=0.074..0.100 rows=59 loops=1)
61	[...] Sort Key: r.dni
62	[...] Sort Method: quicksort Memory: 27kB
63	[...] -> Seq Scan on repartidor r (cost=0.00..1.60 rows=60 width=9) (actual time=0.011..0.018 rows=60 loops=1)
64	[...] -> Index Scan using pk_persona_dni on persona p2 (cost=0.29..0.40 rows=1 width=15) (actual time=0.026..0.026 rows=1 loops=21)
65	[...] Index Cond: ((dni)::text = (d.rdn)::text)
66	[...] -> Hash Join (cost=384.59..542.09 rows=1800 width=9) (actual time=0.432..2.434 rows=3485 loops=21)
67	[...] Hash Cond: (cl.correlativo = b.correlativo)
68	[...] -> Seq Scan on compralocal cl (cost=0.00..117.00 rows=6000 width=4) (actual time=0.003..0.475 rows=4413 loops=21)
69	[...] -> Hash (cost=362.09..362.09 rows=1800 width=17) (actual time=8.979..8.983 rows=4738 loops=1)
70	[...] Buckets: 8192 (originally 2048) Batches: 1 (originally 1) Memory Usage: 305kB
71	[...] -> Hash Join (cost=263.91..362.09 rows=1800 width=17) (actual time=5.104..7.824 rows=4738 loops=1)
72	[...] Hash Cond: (b.correlativo = c.correlativo)
73	[...] -> Seq Scan on boleta b (cost=0.00..84.00 rows=5400 width=13) (actual time=0.013..0.650 rows=5400 loops=1)
74	[...] -> Hash (cost=222.62..222.62 rows=3303 width=4) (actual time=5.064..5.065 rows=8828 loops=1)
75	[...] Buckets: 16384 (originally 4096) Batches: 1 (originally 1) Memory Usage: 439kB
76	[...] -> Seq Scan on compra c (cost=0.00..222.62 rows=3303 width=4) (actual time=0.015..3.460 rows=8828 loops=1)
77	[...] Filter: (EXTRACT(year FROM fecha_hora) < '2021'::numeric)
78	[...] Rows Removed by Filter: 1172
79	Planning Time: 3.675 ms
80	Execution Time: 244.155 ms

Figura 54: Query Plan de la consulta 3 de 10k datos sin índices

5.3.3.4. Ejecución con índices para 10k datos

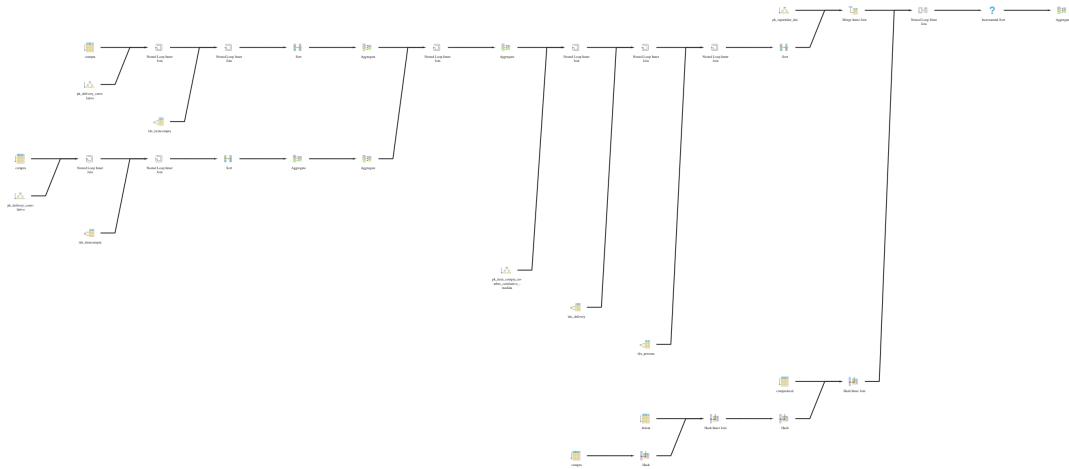


Figura 55: Query Plan Tree de la consulta 3 de 10k datos con índices

1	GroupAggregate (cost=1082.74..1173.08 rows=1 width=30) (actual time=233.615..233.672 rows=9 loops=1)
2	[...] Group Key: r.dni, p2.nombre, i.nombre_producto
3	[...] > Incremental Sort (cost=1082.74..1173.05 rows=2 width=22) (actual time=233.598..233.633 rows=9 loops=1)
4	[...] Sort Key: r.dni, p2.nombre, i.nombre_producto
5	[...] Presorted Key: r.dni
6	[...] Full-sort Groups: 1 Sort Method: quicksort Average Memory: 25kB Peak Memory: 25kB
7	[...] > Nested Loop Semi Join (cost=992.50..1172.96 rows=1 width=22) (actual time=59.355..233.558 rows=9 loops=1)
8	[...] Join Filter: ((d.cdni)::text = (b.dni)::text)
9	[...] Rows Removed by Join Filter: 73178
10	[...] > Nested Loop (cost=622.44..622.90 rows=1 width=31) (actual time=44.160..44.860 rows=21 loops=1)
11	[...] > Merge Join (cost=622.44..622.76 rows=1 width=34) (actual time=44.121..44.413 rows=21 loops=1)
12	[...] Merge Cond: ((d.rdn1)::text = (r.dni)::text)
13	[...] > Sort (cost=619.07..619.07 rows=1 width=25) (actual time=37.440..37.512 rows=21 loops=1)
14	[...] Sort Key: d.rdn1
15	[...] Sort Method: quicksort Memory: 26kB
16	[...] > Nested Loop (cost=614.91..619.06 rows=1 width=25) (actual time=37.173..37.444 rows=21 loops=1)
17	[...] > Nested Loop (cost=614.91..618.98 rows=2 width=11) (actual time=37.152..37.284 rows=37 loops=1)
18	[...] > HashAggregate (cost=614.63..614.64 rows=1 width=7) (actual time=37.057..37.084 rows=15 loops=1)
19	[...] Group Key: (l_1.nombre_producto)::text
20	[...] Batches: 1 Memory Usage: 24kB
21	[...] > Nested Loop (cost=614.56..614.62 rows=1 width=7) (actual time=15.912..36.987 rows=15 loops=1)
22	[...] Join Filter: ((sum(l_1.cantidad)) = (max((sum(i_2.cantidad))))
23	[...] Rows Removed by Join Filter: 149
24	[...] > GroupAggregate (cost=307.26..307.28 rows=1 width=15) (actual time=7.275..7.433 rows=164 loops=1)
25	[...] Group Key: l_1.nombre_producto
26	[...] > Sort (cost=307.26..307.27 rows=1 width=11) (actual time=7.261..7.310 rows=164 loops=1)
27	[...] Sort Key: l_1.nombre_producto
28	[...] Sort Method: quicksort Memory: 33kB
29	[...] > Nested Loop (cost=0.28..307.25 rows=1 width=11) (actual time=0.106..6.969 rows=164 loops=1)
30	[...] > Nested Loop (cost=0.28..307.05 rows=1 width=8) (actual time=0.091..6.333 rows=148 loops=1)
31	[...] > Seq Scan on compra c_1 (cost=0.00..302.75 rows=1 width=4) (actual time=0.055..5.506 rows=411 loops=1)
32	[...] Filter: ((EXTRACT(year FROM fechaHora) = '2021'::numeric) AND (EXTRACT(month FROM fechaHora) = ANY ('(1,2,3,4,5,6)'::numeric[])))
33	[...] Rows Removed by Filter: 9589
34	[...] > Index Only Scan using pk_delivery_correlativo on delivery d_1 (cost=0.28..4.30 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=411)
35	[...] Index Cond: (correlativo = c_1.correlativo)
36	[...] Heap Fetches: 0
37	[...] > Index Scan using idx_itemcompra on itemcompra i_1 (cost=0.00..0.19 rows=2 width=15) (actual time=0.003..0.004 rows=1 loops=148)
38	[...] Index Cond: (correlativo = d_1.correlativo)
39	[...] > Aggregate (cost=307.30..307.31 rows=1 width=8) (actual time=0.179..0.179 rows=1 loops=164)
40	[...] > GroupAggregate (cost=307.26..307.28 rows=1 width=15) (actual time=0.044..0.165 rows=164 loops=164)

41	[...] Group Key: i_2.nombre_producto
42	[...] > Sort (cost=307.26..307.27 rows=1 width=11) (actual time=0.043..0.087 rows=164 loops=164)
43	[...] Sort Key: i_2.nombre_producto
44	[...] Sort Method: quicksort Memory: 33kB
45	[...] > Nested Loop (cost=0.28..307.25 rows=1 width=11) (actual time=0.081..6.705 rows=164 loops=1)
46	[...] > Nested Loop (cost=0.28..307.05 rows=1 width=8) (actual time=0.073..5.804 rows=148 loops=1)
47	[...] > Seq Scan on compra c_2 (cost=0.00..302.75 rows=1 width=4) (actual time=0.049..4.885 rows=411 loops=1)
48	[...] Filter: ((EXTRACT(year FROM fecha_hora) = '2021'::numeric) AND (EXTRACT(month FROM fecha_hora) = ANY ('1,2,3,4,5,6')::numeric[]))
49	[...] Rows Removed by Filter: 9589
50	[...] > Index Only Scan using pk_delivery_correlativo on delivery d_2 (cost=0.28..4.30 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=411)
51	[...] Index Cond: (correlativo = c_2.correlativo)
52	[...] Heap Fetches: 0
53	[...] > Index Scan using idx_itemcompra on itemcompra i_2 (cost=0.00..0.19 rows=2 width=15) (actual time=0.003..0.005 rows=1 loops=148)
54	[...] Index Cond: (correlativo = d_2.correlativo)
55	[...] > Index Only Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i (cost=0.29..4.32 rows=2 width=11) (actual time=0.011..0.012 rows=2 loops=15)
56	[...] Index Cond: (nombre_producto = (i_1.nombre_producto)::text)
57	[...] Heap Fetches: 0
58	[...] > Index Scan using idx_delivery on delivery d (cost=0.00..0.04 rows=1 width=22) (actual time=0.004..0.004 rows=1 loops=37)
59	[...] Index Cond: (correlativo = i.correlativo)
60	[...] > Sort (cost=3.37..3.52 rows=60 width=9) (actual time=6.664..6.685 rows=59 loops=1)
61	[...] Sort Key: r.dni
62	[...] Sort Method: quicksort Memory: 27kB
63	[...] > Seq Scan on repartidor r (cost=0.00..1.60 rows=60 width=9) (actual time=0.030..0.049 rows=60 loops=1)
64	[...] > Index Scan using idx_persona on persona p_2 (cost=0.00..0.15 rows=1 width=15) (actual time=0.012..0.012 rows=1 loops=21)
65	[...] Index Cond: ((dni)::text = (d.dni)::text)
66	[...] > Hash Join (cost=370.06..527.56 rows=1800 width=9) (actual time=0.520..8.594 rows=3485 loops=21)
67	[...] Hash Cond: (cl.correlativo = b.correlativo)
68	[...] > Seq Scan on compralocal cl (cost=0.00..117.00 rows=6000 width=4) (actual time=0.003..1.931 rows=4413 loops=21)
69	[...] > Hash (cost=347.56..347.56 rows=1800 width=17) (actual time=10.796..10.800 rows=4738 loops=1)
70	[...] Buckets: 8192 (originally 2048) Batches: 1 (originally 1) Memory Usage: 305kB
71	[...] > Hash Join (cost=249.38..347.56 rows=1800 width=17) (actual time=6.016..9.778 rows=4738 loops=1)
72	[...] Hash Cond: (b.correlativo = c.correlativo)
73	[...] > Seq Scan on boleta b (cost=0.00..84.00 rows=5400 width=13) (actual time=0.008..0.594 rows=5400 loops=1)
74	[...] > Hash (cost=211.25..211.25 rows=3050 width=4) (actual time=5.982..5.983 rows=8828 loops=1)
75	[...] Buckets: 16384 (originally 4096) Batches: 1 (originally 1) Memory Usage: 439kB
76	[...] > Seq Scan on compra c (cost=0.00..211.25 rows=3050 width=4) (actual time=0.142..4.368 rows=8828 loops=1)
77	[...] Filter: (EXTRACT(year FROM fecha_hora) < '2021'::numeric)
78	[...] Rows Removed by Filter: 1172
79	Planning Time: 18.525 ms
80	Execution Time: 234.198 ms

Figura 56: Query Plan de la consulta 3 de 10k datos con índices

5.3.3.5. Ejecución sin índices para 100k datos

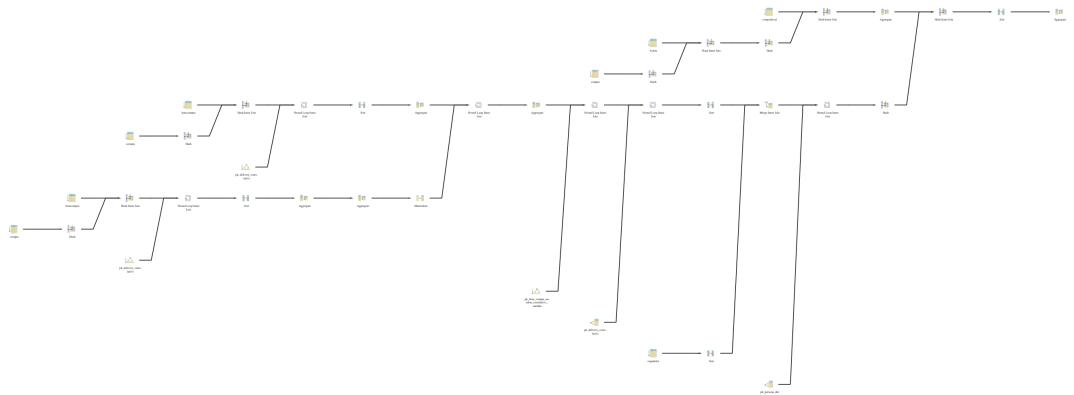


Figura 57: Query Plan Tree de la consulta 3 de 100k datos sin índices

1	GroupAggregate (cost=16158.33..16158.36 rows=1 width=32) (actual time=855.463..855.493 rows=1 loops=1)
2	[...] Group Key: r.dni, p2.nombre, i.nombre_producto
3	[...] > Sort (cost=16158.33..16158.34 rows=1 width=24) (actual time=855.448..855.478 rows=1 loops=1)
4	[...] Sort Key: r.dni, p2.nombre, i.nombre_producto
5	[...] Sort Method: quicksort Memory: 25kB
6	[...] > Hash Join (cost=15910.91..16158.32 rows=1 width=24) (actual time=811.818..855.460 rows=1 loops=1)
7	[...] Hash Cond: ((b.dni)::text = (d.dni)::text)
8	[...] > HashAggregate (cost=5373.98..5553.91 rows=17993 width=9) (actual time=444.704..478.466 rows=34620 loops=1)
9	[...] Group Key: (b.dni)::text
10	[...] Batches: 5 Memory Usage: 4145kB Disk Usage: 248kB
11	[...] > Hash Join (cost=3838.23..5329.00 rows=17993 width=9) (actual time=281.051..401.105 rows=47721 loops=1)
12	[...] Hash Cond: (cl.correlativo = b.correlativo)
13	[...] > Seq Scan on compralocal cl (cost=0.00..1085.88 rows=59980 width=4) (actual time=0.013..35.344 rows=60000 loops=1)
14	[...] > Hash (cost=3613.32..3613.32 rows=17993 width=17) (actual time=280.974..280.979 rows=47721 loops=1)
15	[...] Buckets: 65536 (originally 32768) Batches: 1 (originally 1) Memory Usage: 2936kB
16	[...] > Hash Join (cost=2639.81..3613.32 rows=17993 width=17) (actual time=94.072..251.538 rows=47721 loops=1)
17	[...] Hash Cond: (b.correlativo = c.correlativo)
18	[...] > Seq Scan on boleta b (cost=0.00..831.80 rows=53980 width=13) (actual time=0.008..6.200 rows=54000 loops=1)
19	[...] > Hash (cost=2226.38..2226.38 rows=33075 width=4) (actual time=93.750..93.752 rows=88298 loops=1)
20	[...] Buckets: 131072 (originally 65536) Batches: 2 (originally 1) Memory Usage: 3073kB
21	[...] > Seq Scan on compra c (cost=0.00..2226.38 rows=33075 width=4) (actual time=0.021..55.737 rows=88298 loops=1)
22	[...] Filter: (EXTRACT(year FROM fecha_hora) < '2021'::numeric)
23	[...] Rows Removed by Filter: 11702
24	[...] > Hash (cost=10536.90..10536.90 rows=2 width=33) (actual time=346.219..346.240 rows=2 loops=1)
25	[...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
26	[...] > Nested Loop (cost=10535.73..10536.90 rows=2 width=33) (actual time=346.189..346.230 rows=2 loops=1)
27	[...] > Merge Join (cost=10535.31..10535.84 rows=2 width=36) (actual time=346.141..346.168 rows=2 loops=1)
28	[...] Merge Cond: ((d.rdn1)::text = (r.dni)::text)
29	[...] > Sort (cost=10529.99..10529.99 rows=2 width=27) (actual time=345.652..345.669 rows=2 loops=1)
30	[...] Sort Key: d.rdn1
31	[...] Sort Method: quicksort Memory: 25kB
32	[...] > Nested Loop (cost=10525.55..10529.98 rows=2 width=27) (actual time=345.623..345.655 rows=2 loops=1)
33	[...] > Nested Loop (cost=10525.26..10529.32 rows=2 width=13) (actual time=345.601..345.619 rows=3 loops=1)
34	[...] > HashAggregate (cost=10524.84..10524.85 rows=1 width=9) (actual time=345.506..345.521 rows=1 loops=1)
35	[...] Group Key: (i_1.nombre_producto)::text
36	[...] Batches: 1 Memory Usage: 24kB
37	[...] > Nested Loop (cost=10524.35..10524.84 rows=1 width=9) (actual time=344.159..345.491 rows=1 loops=1)
38	[...] Join Filter: ((sum(l_1.cantidad)) = (max((sum(l_2.cantidad)))))
39	[...] Rows Removed by Join Filter: 1613
40	[...] > GroupAggregate (cost=5262.01..5262.20 rows=11 width=17) (actual time=224.031..225.062 rows=1614 loops=1)

41	[...] Group Key: i_1.nombre_producto
42	[...] > Sort (cost=5262.01..5262.04 rows=11 width=13) (actual time=224.013..224.260 rows=1627 loops=1)
43	[...] Sort Key: i_1.nombre_producto
44	[...] Sort Method: quicksort Memory: 139kB
45	[...] > Nested Loop (cost=3219.10..5261.82 rows=11 width=13) (actual time=66.279..210.792 rows=1627 loops=1)
46	[...] -> Hash Join (cost=3218.81..5257.14 rows=15 width=21) (actual time=66.225..155.992 rows=4139 loops=1)
47	[...] Hash Cond: (i_1.correlativo = c_1.correlativo)
48	[...] -> Seq Scan on itemcompra i_1 (cost=0.00..1782.09 rows=97609 width=17) (actual time=0.025..36.796 rows=100000 loops=1)
49	[...] -> Hash (cost=3218.62..3218.62 rows=15 width=4) (actual time=66.165..66.166 rows=4058 loops=1)
50	[...] Buckets: 4096 (originally 1024) Batches: 1 (originally 1) Memory Usage: 175kB
51	[...] -> Seq Scan on compra c_1 (cost=0.00..3218.62 rows=15 width=4) (actual time=0.027..64.583 rows=4058 loops=1)
52	[...] Filter: ((EXTRACT(year FROM fechaHora) = '2021'::numeric) AND (EXTRACT(month FROM fechaHora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
53	[...] Rows Removed by Filter: 95942
54	[...] -> Index Only Scan using pk_delivery_correlativo on delivery d_1 (cost=0.29..0.31 rows=1 width=4) (actual time=0.009..0.009 rows=0 loops=4139)
55	[...] Index Cond: (correlativo = i_1.correlativo)
56	[...] Heap Fetches: 0
57	[...] -> Materialize (cost=5262.34..5262.36 rows=1 width=8) (actual time=0.074..0.074 rows=1 loops=1614)
58	[...] -> Aggregate (cost=5262.34..5262.35 rows=1 width=8) (actual time=119.771..119.777 rows=1 loops=1)
59	[...] -> GroupAggregate (cost=5262.01..5262.20 rows=11 width=17) (actual time=118.711..119.631 rows=1614 loops=1)
60	[...] Group Key: i_2.nombre_producto
61	[...] -> Sort (cost=5262.01..5262.04 rows=11 width=13) (actual time=118.690..118.863 rows=1627 loops=1)
62	[...] Sort Key: i_2.nombre_producto
63	[...] Sort Method: quicksort Memory: 139kB
64	[...] -> Nested Loop (cost=3219.10..5261.82 rows=11 width=13) (actual time=70.729..115.151 rows=1627 loops=1)
65	[...] -> Hash Join (cost=3218.81..5257.14 rows=15 width=21) (actual time=70.677..101.866 rows=4139 loops=1)
66	[...] Hash Cond: (i_2.correlativo = c_2.correlativo)
67	[...] -> Seq Scan on itemcompra i_2 (cost=0.00..1782.09 rows=97609 width=17) (actual time=0.026..11.836 rows=100000 loops=1)
68	[...] -> Hash (cost=3218.62..3218.62 rows=15 width=4) (actual time=62.054..62.055 rows=4058 loops=1)
69	[...] Buckets: 4096 (originally 1024) Batches: 1 (originally 1) Memory Usage: 175kB
70	[...] -> Seq Scan on compra c_2 (cost=0.00..3218.62 rows=15 width=4) (actual time=0.549..60.252 rows=4058 loops=1)
71	[...] Filter: ((EXTRACT(year FROM fechaHora) = '2021'::numeric) AND (EXTRACT(month FROM fechaHora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
72	[...] Rows Removed by Filter: 95942
73	[...] -> Index Only Scan using pk_delivery_correlativo on delivery d_2 (cost=0.29..0.31 rows=1 width=4) (actual time=0.003..0.003 rows=0 loops=4139)
74	[...] Index Cond: (correlativo = i_2.correlativo)
75	[...] Heap Fetches: 0
76	[...] -> Index Only Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i (cost=0.42..4.45 rows=2 width=13) (actual time=0.087..0.088 rows=3 loops=1)
77	[...] Index Cond: (nombre_producto = (i_1.nombre_producto)::text)
78	[...] Heap Fetches: 0
79	[...] -> Index Scan using pk_delivery_correlativo on delivery d (cost=0.29..0.33 rows=1 width=22) (actual time=0.009..0.010 rows=1 loops=3)
80	[...] Index Cond: (correlativo = i.correlativo)
81	[...] -> Sort (cost=5.32..5.57 rows=100 width=9) (actual time=0.471..0.477 rows=70 loops=1)
82	[...] Sort Key: r.rdn
83	[...] Sort Method: quicksort Memory: 29kB
84	[...] -> Seq Scan on repartidor r (cost=0.00..2.00 rows=100 width=9) (actual time=0.009..0.021 rows=100 loops=1)
85	[...] -> Index Scan using pk_persona_dni on persona p2 (cost=0.42..0.53 rows=1 width=15) (actual time=0.026..0.026 rows=1 loops=2)
86	[...] Index Cond: ((dni)::text = (d.rdn)::text)
87	Planning Time: 7.963 ms
88	Execution Time: 857.435 ms

Figura 58: Query Plan de la consulta 3 de 100k datos sin índices

5.3.3.6. Ejecución con índices para 100k datos

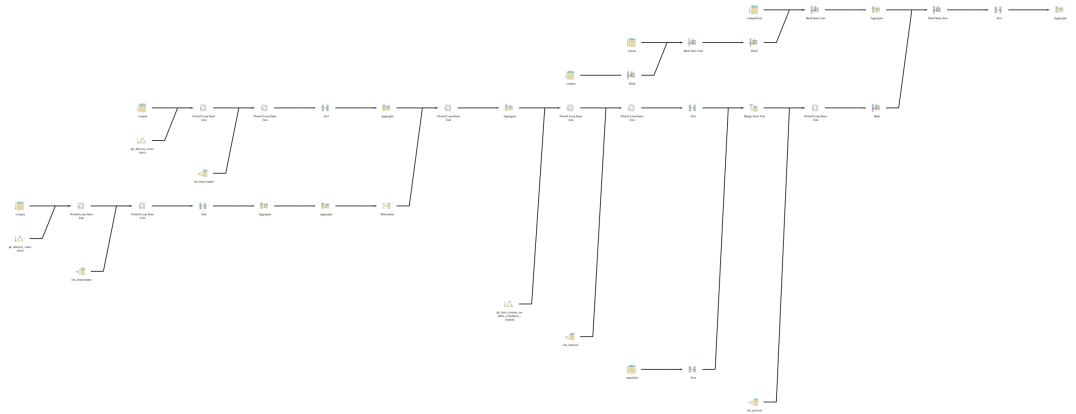


Figura 59: Query Plan Tree de la consulta 3 de 100k datos con índices

1	GroupAggregate (cost=12181.88..12181.90 rows=1 width=32) (actual time=542.086..542.117 rows=1 loops=1)
2	[...] Group Key: r.dni, p2.nombre, i.nombre_producto
3	[...] > Sort (cost=12181.88..12181.88 rows=1 width=24) (actual time=542.076..542.107 rows=1 loops=1)
4	[...] Sort Key: r.dni, p2.nombre, i.nombre_producto
5	[...] Sort Method: quicksort Memory: 25kB
6	[...] > Hash Join (cost=11935.00..12181.87 rows=1 width=24) (actual time=528.361..541.914 rows=1 loops=1)
7	[...] Hash Cond: ((b.dni)::text = (d.dni)::text)
8	[...] > HashAggregate (cost=5366.37..5545.90 rows=17953 width=9) (actual time=384.356..399.857 rows=34620 loops=1)
9	[...] Group Key: (b.dni)::text
10	[...] Batches: 5 Memory Usage: 4145kB Disk Usage: 248kB
11	[...] > Hash Join (cost=3831.62..5321.49 rows=17953 width=9) (actual time=268.878..331.781 rows=47721 loops=1)
12	[...] Hash Cond: (cl.correlativo = b.correlativo)
13	[...] > Seq Scan on compralocal cl (cost=0.00..1085.52 rows=59952 width=4) (actual time=0.011..12.329 rows=60000 loops=1)
14	[...] > Hash (cost=3607.21..3607.21 rows=17953 width=17) (actual time=268.816..268.821 rows=47721 loops=1)
15	[...] Buckets: 65536 (originally 32768) Batches: 1 (originally 1) Memory Usage: 2936kB
16	[...] > Hash Join (cost=2635.21..3607.21 rows=17953 width=17) (actual time=113.313..226.550 rows=47721 loops=1)
17	[...] Hash Cond: (b.correlativo = c.correlativo)
18	[...] > Seq Scan on boleta b (cost=0.00..830.60 rows=53860 width=13) (actual time=0.005..9.940 rows=54000 loops=1)
19	[...] > Hash (cost=2222.77..2222.77 rows=32995 width=4) (actual time=112.652..112.654 rows=88298 loops=1)
20	[...] Buckets: 131072 (originally 65536) Batches: 2 (originally 1) Memory Usage: 3073kB
21	[...] > Seq Scan on compra c (cost=0.00..2222.77 rows=32995 width=4) (actual time=0.021..62.500 rows=88298 loops=1)
22	[...] Filter: (EXTRACT(year FROM fecha_hora) < '2021':numeric)
23	[...] Rows Removed by Filter: 11702
24	[...] > Hash (cost=6568.61..6568.61 rows=2 width=33) (actual time=138.096..138.118 rows=2 loops=1)
25	[...] Buckets: 1024 Batches: 1 Memory Usage: 9kB
26	[...] > Nested Loop (cost=6567.82..6568.61 rows=2 width=33) (actual time=138.021..138.057 rows=2 loops=1)
27	[...] > Merge Join (cost=6567.82..6568.35 rows=2 width=36) (actual time=138.006..138.033 rows=2 loops=1)
28	[...] Merge Cond: ((d.rdn1)::text = (r.dni)::text)
29	[...] > Sort (cost=6562.50..6562.51 rows=2 width=27) (actual time=137.869..137.887 rows=2 loops=1)
30	[...] Sort Key: d.rdn1
31	[...] Sort Method: quicksort Memory: 25kB
32	[...] > Nested Loop (cost=6558.34..6562.49 rows=2 width=27) (actual time=137.835..137.865 rows=2 loops=1)
33	[...] > Nested Loop (cost=6558.34..6562.41 rows=2 width=13) (actual time=137.812..137.830 rows=3 loops=1)
34	[...] > HashAggregate (cost=6557.92..6557.93 rows=1 width=9) (actual time=137.696..137.709 rows=1 loops=1)
35	[...] Group Key: (i_1.nombre_producto)::text
36	[...] Batches: 1 Memory Usage: 24kB
37	[...] > Nested Loop (cost=6557.43..6557.92 rows=1 width=9) (actual time=136.346..137.695 rows=1 loops=1)
38	[...] Join Filter: ((sum(l_1.cantidad)) = (max((sum(i_2.cantidad)))))
39	[...] Rows Removed by Join Filter: 1613
40	[...] > GroupAggregate (cost=3278.55..3278.74 rows=11 width=17) (actual time=67.913..68.962 rows=1614 loops=1)

41	[...] Group Key: i_1.nombre_producto
42	[...] > Sort (cost=3278.55..3278.58 rows=11 width=13) (actual time=67.898..68.153 rows=1627 loops=1)
43	[...] Sort Key: i_1.nombre_producto
44	[...] Sort Method: quicksort Memory: 139kB
45	[...] > Nested Loop (cost=0.29..3278.36 rows=11 width=13) (actual time=0.197..63.038 rows=1627 loops=1)
46	[...] -> Nested Loop (cost=0.29..3277.24 rows=6 width=8) (actual time=0.066..52.743 rows=1600 loops=1)
47	[...] -> Seq Scan on compra_c_1 (cost=0.00..3212.62 rows=15 width=4) (actual time=0.027..43.624 rows=4058 loops=1)
48	[...] Filter: ((EXTRACT(year FROM fecha_hora) = '2021'::numeric) AND (EXTRACT(month FROM fecha_hora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
49	[...] Rows Removed by Filter: 95942
50	[...] -> Index Only Scan using pk_delivery_correlativo on delivery d_1 (cost=0.29..4.31 rows=1 width=4) (actual time=0.002..0.002 rows=0 loops=4058)
51	[...] Index Cond: (correlativo = c_1.correlativo)
52	[...] Heap Fetches: 0
53	[...] -> Index Scan using idx_itemcompra on itemcompra i_1 (cost=0.00..0.17 rows=2 width=17) (actual time=0.005..0.006 rows=1 loops=1600)
54	[...] Index Cond: (correlativo = d_1.correlativo)
55	[...] -> Materialize (cost=3278.88..3278.91 rows=1 width=8) (actual time=0.042..0.042 rows=1 loops=1614)
56	[...] -> Aggregate (cost=3278.88..3278.89 rows=1 width=8) (actual time=68.099..68.105 rows=1 loops=1)
57	[...] -> GroupAggregate (cost=3278.55..3278.74 rows=11 width=17) (actual time=67.082..67.959 rows=1614 loops=1)
58	[...] Group Key: i_2.nombre_producto
59	[...] > Sort (cost=3278.55..3278.58 rows=11 width=13) (actual time=67.068..67.207 rows=1627 loops=1)
60	[...] Sort Key: i_2.nombre_producto
61	[...] Sort Method: quicksort Memory: 139kB
62	[...] > Nested Loop (cost=0.29..3278.36 rows=11 width=13) (actual time=0.150..64.224 rows=1627 loops=1)
63	[...] -> Nested Loop (cost=0.29..3277.24 rows=6 width=8) (actual time=0.044..56.055 rows=1600 loops=1)
64	[...] -> Seq Scan on compra_c_2 (cost=0.00..3212.62 rows=15 width=4) (actual time=0.023..39.029 rows=4058 loops=1)
65	[...] Filter: ((EXTRACT(year FROM fecha_hora) = '2021'::numeric) AND (EXTRACT(month FROM fecha_hora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
66	[...] Rows Removed by Filter: 95942
67	[...] -> Index Only Scan using pk_delivery_correlativo on delivery d_2 (cost=0.29..4.31 rows=1 width=4) (actual time=0.003..0.003 rows=0 loops=4058)
68	[...] Index Cond: (correlativo = c_2.correlativo)
69	[...] Heap Fetches: 0
70	[...] -> Index Scan using idx_itemcompra on itemcompra i_2 (cost=0.00..0.17 rows=2 width=17) (actual time=0.004..0.004 rows=1 loops=1600)
71	[...] Index Cond: (correlativo = d_2.correlativo)
72	[...] -> Index Only Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i (cost=0.42..4.45 rows=2 width=13) (actual time=0.107..0.110 rows=3 loops=1)
73	[...] Index Cond: (nombre_producto = (i_1.nombre_producto)::text)
74	[...] Heap Fetches: 0
75	[...] -> Index Scan using idx_delivery on delivery d (cost=0.00..0.04 rows=1 width=22) (actual time=0.009..0.009 rows=1 loops=3)
76	[...] Index Cond: (correlativo = i.correlativo)
77	[...] -> Sort (cost=5.32..5.57 rows=100 width=9) (actual time=0.117..0.123 rows=70 loops=1)
78	[...] Sort Key: r.dni
79	[...] Sort Method: quicksort Memory: 29kB
80	[...] > Seq Scan on repartidor r (cost=0.00..2.00 rows=100 width=9) (actual time=0.010..0.022 rows=100 loops=1)
81	[...] -> Index Scan using idx_persona on persona p2 (cost=0.00..0.13 rows=1 width=15) (actual time=0.008..0.008 rows=1 loops=2)
82	[...] Index Cond: ((dni)::text = (d.rdn)::text)
83	Planning Time: 19.257 ms
84	Execution Time: 543.809 ms

Figura 60: Query Plan de la consulta 3 de 100k datos con índices

5.3.3.7. Ejecución sin índices para 1m datos

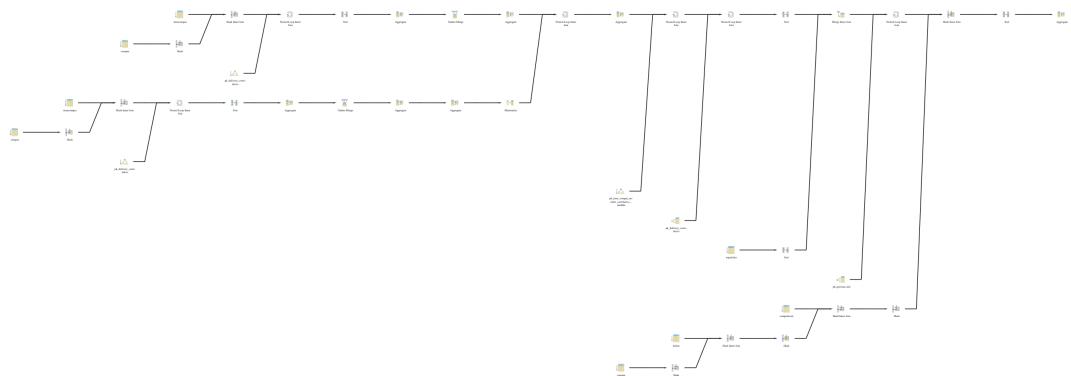


Figura 61: Query Plan Tree de la consulta 3 de 1m datos sin índices

1	GroupAggregate (cost=109663.35..109663.40 rows=2 width=30) (actual time=4704.373..4704.707 rows=1 loops=1)
2	[...] Group Key: r.dni, p2.nombre, i.nombre_producto
3	[...] > Sort (cost=109663.35..109663.36 rows=2 width=22) (actual time=4704.357..4704.691 rows=1 loops=1)
4	[...] Sort Key: r.dni, p2.nombre, i.nombre_producto
5	[...] Sort Method: quicksort Memory: 25kB
6	[...] > Hash Semi Join (cost=108780.93..109663.34 rows=2 width=22) (actual time=4704.199..4704.648 rows=1 loops=1)
7	[...] Hash Cond: ((d.cdn)::text = (b.dni)::text)
8	[...] > Nested Loop (cost=38352.75..38354.13 rows=2 width=31) (actual time=1435.764..1436.265 rows=4 loops=1)
9	[...] > Merge Join (cost=38352.33..38353.06 rows=2 width=34) (actual time=1435.718..1436.098 rows=4 loops=1)
10	[...] Merge Cond: ((d.rdn)::text = (r.dni)::text)
11	[...] > Sort (cost=38344.94..38344.94 rows=2 width=25) (actual time=1435.504..1435.830 rows=4 loops=1)
12	[...] Sort Key: d.rdn
13	[...] Sort Method: quicksort Memory: 25kB
14	[...] > Nested Loop (cost=38340.40..38344.93 rows=2 width=25) (actual time=1435.439..1435.816 rows=4 loops=1)
15	[...] > Nested Loop (cost=38339.98..38344.01 rows=2 width=11) (actual time=1435.401..1435.725 rows=5 loops=1)
16	[...] > HashAggregate (cost=38339.55..38339.56 rows=1 width=7) (actual time=1435.263..1435.582 rows=1 loops=1)
17	[...] Group Key: (l_1.nombre_producto)::text
18	[...] Batches: 1 Memory Usage: 24kB
19	[...] > Nested Loop (cost=38323.07..38339.55 rows=1 width=7) (actual time=1366.463..1435.547 rows=1 loops=1)
20	[...] Join Filter: ((sum(l_1.cantidad)) = (max((sum(l_2.cantidad)))))
21	[...] Rows Removed by Join Filter: 16057
22	[...] > Finalize GroupAggregate (cost=19154.03..19167.59 rows=116 width=15) (actual time=759.811..849.319 rows=16058 loops=1)
23	[...] Group Key: l_1.nombre_producto
24	[...] > Gather Merge (cost=19154.03..19165.95 rows=96 width=15) (actual time=759.778..830.210 rows=16129 loops=1)
25	[...] Workers Planned: 2
26	[...] Workers Launched: 2
27	[...] > Partial GroupAggregate (cost=18154.00..18154.84 rows=48 width=15) (actual time=663.123..672.325 rows=5376 loops=3)
28	[...] Group Key: l_1.nombre_producto
29	[...] > Sort (cost=18154.00..18154.12 rows=48 width=11) (actual time=663.081..664.604 rows=5398 loops=3)
30	[...] Sort Key: l_1.nombre_producto
31	[...] Sort Method: quicksort Memory: 487kB
32	[...] Worker 0: Sort Method: quicksort Memory: 402kB
33	[...] Worker 1: Sort Method: quicksort Memory: 450kB
34	[...] > Nested Loop (cost=0.42..18152.66 rows=48 width=11) (actual time=65.327..633.864 rows=5398 loops=3)
35	[...] > Nested Loop (cost=0.42..18148.40 rows=25 width=8) (actual time=65.272..464.909 rows=5328 loops=3)
36	[...] > Parallel Seq Scan on compra c_1 (cost=0.00..17888.00 rows=62 width=4) (actual time=63.992..361.617 rows=13192 loops=3)
37	[...] Filter: ((EXTRACT(year FROM fecha_hora) = '2021'::numeric) AND (EXTRACT(month FROM fecha_hora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
38	[...] Rows Removed by Filter: 320142
39	[...] > Index Only Scan using pk_delivery_correlativo on delivery d_1 (cost=0.42..4.20 rows=1 width=4) (actual time=0.007..0.007 rows=0 loops=39575)
40	[...] Index Cond: (correlativo = c_1.correlativo)

41	[...] Heap Fetches: 0
42	[...] -> Index Scan using index_item_compra_correlativo on itemcompra i_1 (cost=0.00..0.15 rows=2 width=15) (actual time=0.024..0.030 rows=1 loops=15983)
43	[...] Index Cond: (correlativo = d_1.correlativo)
44	[...] Rows Removed by Index Recheck: 0
45	[...] -> Materialize (cost=19169.04..19169.06 rows=1 width=8) (actual time=0.036..0.036 rows=1 loops=16058)
46	[...] -> Aggregate (cost=19169.04..19169.05 rows=1 width=8) (actual time=576.935..577.047 rows=1 loops=1)
47	[...] -> Finalize GroupAggregate (cost=19154.03..19167.59 rows=116 width=15) (actual time=495.607..574.557 rows=16058 loops=1)
48	[...] Group Key: i_2.nombre_producto
49	[...] -> Gather Merge (cost=19154.03..19165.95 rows=96 width=15) (actual time=495.595..564.399 rows=16139 loops=1)
50	[...] Workers Planned: 2
51	[...] Workers Launched: 2
52	[...] -> Partial GroupAggregate (cost=18154.00..18154.84 rows=48 width=15) (actual time=446.361..453.083 rows=5380 loops=3)
53	[...] Group Key: i_2.nombre_producto
54	[...] -> Sort (cost=18154.00..18154.12 rows=48 width=11) (actual time=446.324..447.465 rows=5398 loops=3)
55	[...] Sort Key: i_2.nombre_producto
56	[...] Sort Method: quicksort Memory: 494kB
57	[...] Worker 0: Sort Method: quicksort Memory: 504kB
58	[...] Worker 1: Sort Method: quicksort Memory: 246kB
59	[...] -> Nested Loop (cost=0.42..18152.66 rows=48 width=11) (actual time=14.080..425.058 rows=5398 loops=3)
60	[...] -> Nested Loop (cost=0.42..18148.40 rows=25 width=8) (actual time=13.938..349.922 rows=5328 loops=3)
61	[...] -> Parallel Seq Scan on compra c_2 (cost=0.00..17888.00 rows=62 width=4) (actual time=13.807..264.066 rows=13192 loops=3)
62	[...] Filter: ((EXTRACT(year FROM fechaHora) = '2021'::numeric) AND (EXTRACT(month FROM fechaHora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
63	[...] Rows Removed by Filter: 320142
64	[...] -> Index Only Scan using pk_delivery_correlativo on delivery d_2 (cost=0.42..4.20 rows=1 width=4) (actual time=0.006..0.006 rows=0 loops=39575)
65	[...] Index Cond: (correlativo = c_2.correlativo)
66	[...] Heap Fetches: 0
67	[...] -> Index Scan using index_item_compra_correlativo on itemcompra i_2 (cost=0.00..0.15 rows=2 width=15) (actual time=0.010..0.013 rows=1 loops=15983)
68	[...] Index Cond: (correlativo = d_2.correlativo)
69	[...] Rows Removed by Index Recheck: 0
70	[...] -> Index Only Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i (cost=0.42..4.43 rows=2 width=11) (actual time=0.126..0.128 rows=5 loops=1)
71	[...] Index Cond: (nombre_producto = (i_1.nombre_producto)::text)
72	[...] Heap Fetches: 0
73	[...] -> Index Scan using pk_delivery_correlativo on delivery d (cost=0.42..0.46 rows=1 width=22) (actual time=0.016..0.016 rows=1 loops=5)
74	[...] Index Cond: (correlativo = i.correlativo)
75	[...] -> Sort (cost=7.39..7.74 rows=140 width=9) (actual time=0.204..0.222 rows=122 loops=1)
76	[...] Sort Key: r.rdnI
77	[...] Sort Method: quicksort Memory: 31kB
78	[...] -> Seq Scan on repartidor r (cost=0.00..2.40 rows=140 width=9) (actual time=0.011..0.030 rows=140 loops=1)
79	[...] -> Index Scan using pk_persona_dni on persona p2 (cost=0.42..0.54 rows=1 width=15) (actual time=0.036..0.036 rows=1 loops=4)
80	[...] Index Cond: ((dnl)::text = (d.rdnI)::text)

81	[...] > Hash (cost=67299.62..67299.62 rows=179965 width=9) (actual time=3133.096..3133.102 rows=478194 loops=1)
82	[...] Buckets: 131072 (originally 131072) Batches: 8 (originally 4) Memory Usage: 3388kB
83	[...] > Hash Join (cost=47549.31..67299.62 rows=179965 width=9) (actual time=2082.767..2936.446 rows=478194 loops=1)
84	[...] Hash Cond: (cl.correlativo = b.correlativo)
85	[...] > Seq Scan on compralocal cl (cost=0.00..9966.48 rows=599248 width=4) (actual time=0.022..110.831 rows=600000 loops=1)
86	[...] > Hash (cost=44244.75..44244.75 rows=179965 width=17) (actual time=2082.317..2082.322 rows=478194 loops=1)
87	[...] Buckets: 65536 (originally 65536) Batches: 8 (originally 4) Memory Usage: 3585kB
88	[...] > Hash Join (cost=27933.56..44244.75 rows=179965 width=17) (actual time=800.321..1886.650 rows=478194 loops=1)
89	[...] Hash Cond: (b.correlativo = c.correlativo)
90	[...] > Seq Scan on boleta b (cost=0.00..8317.95 rows=539895 width=13) (actual time=0.014..139.438 rows=540000 loops=1)
91	[...] > Hash (cost=22467.96..22467.96 rows=333088 width=4) (actual time=799.575..799.576 rows=885256 loops=1)
92	[...] Buckets: 131072 (originally 131072) Batches: 16 (originally 8) Memory Usage: 3073kB
93	[...] > Seq Scan on compra c (cost=0.00..22467.96 rows=333088 width=4) (actual time=0.036..494.762 rows=885256 loops=1)
94	[...] Filter: (EXTRACT(year FROM fechaHora) < '2021'::numeric)
95	[...] Rows Removed by Filter: 114744
96	Planning Time: 4.966 ms
97	JIT:
98	[...] Functions: 165
99	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true
100	[...] Timing: Generation 52.295 ms, Inlining 0.000 ms, Optimization 16.305 ms, Emission 214.177 ms, Total 282.778 ms
101	Execution Time: 4732.474 ms

Figura 62: Query Plan de la consulta 3 de 1m datos sin índices

5.3.3.8. Ejecución con índices para 1m datos

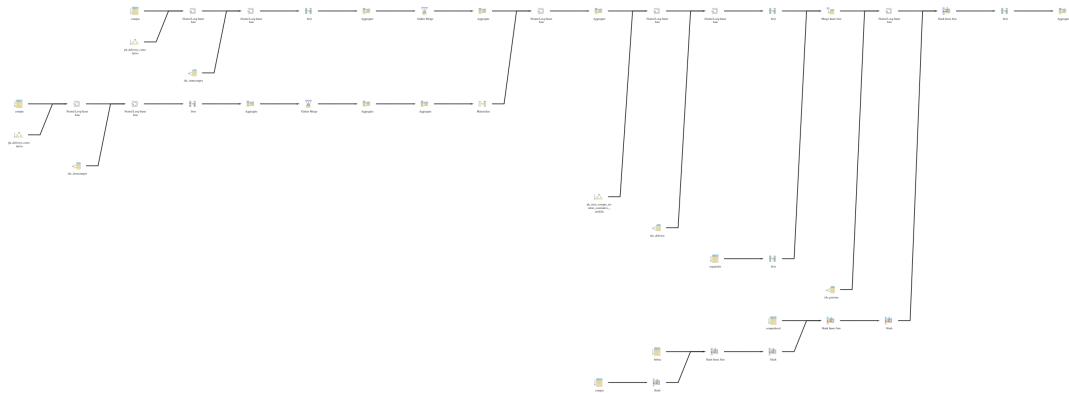


Figura 63: Query Plan Tree de la consulta 3 de 1m datos con índices

1	GroupAggregate (cost=109598.98..109599.03 rows=2 width=30) (actual time=4633.056..4633.353 rows=1 loops=1)
2	[...] Group Key: r.dni, p2.nombre, i.nombre_producto
3	[...] > Sort (cost=109598.98..109598.99 rows=2 width=22) (actual time=4633.041..4633.337 rows=1 loops=1)
4	[...] Sort Key: r.dni, p2.nombre, i.nombre_producto
5	[...] Sort Method: quicksort Memory: 25kB
6	[...] > Hash Semi Join (cost=108716.99..109598.97 rows=2 width=22) (actual time=4632.879..4633.302 rows=1 loops=1)
7	[...] Hash Cond: ((d.cdni)::text = (b.dni)::text)
8	[...] > Nested Loop (cost=38321.61..38322.57 rows=2 width=31) (actual time=1425.502..1425.963 rows=4 loops=1)
9	[...] > Merge Join (cost=38321.61..38322.34 rows=2 width=34) (actual time=1425.454..1425.802 rows=4 loops=1)
10	[...] Merge Cond: ((d.rdn1)::text = (r.dni)::text)
11	[...] > Sort (cost=38314.22..38314.22 rows=2 width=25) (actual time=1425.260..1425.549 rows=4 loops=1)
12	[...] Sort Key: d.rdn1
13	[...] Sort Method: quicksort Memory: 25kB
14	[...] > Nested Loop (cost=38310.10..38314.21 rows=2 width=25) (actual time=1425.179..1425.517 rows=4 loops=1)
15	[...] > Nested Loop (cost=38310.10..38314.13 rows=2 width=11) (actual time=1425.145..1425.434 rows=5 loops=1)
16	[...] > HashAggregate (cost=38309.68..38309.69 rows=1 width=7) (actual time=1425.047..1425.330 rows=1 loops=1)
17	[...] Group Key: (l_1.nombre_producto)::text
18	[...] Batches: 1 Memory Usage: 24kB
19	[...] > Nested Loop (cost=38292.90..38309.67 rows=1 width=7) (actual time=1362.332..1425.295 rows=1 loops=1)
20	[...] Join Filter: ((sum(l_1.cantidad)) = (max((sum(l_2.cantidad)))))
21	[...] Rows Removed by Join Filter: 16057
22	[...] > Finalize GroupAggregate (cost=19138.80..19152.63 rows=117 width=15) (actual time=813.190..892.607 rows=16058 loops=1)
23	[...] Group Key: l_1.nombre_producto
24	[...] > Gather Merge (cost=19138.80..19150.97 rows=98 width=15) (actual time=813.154..882.431 rows=16138 loops=1)
25	[...] Workers Planned: 2
26	[...] Workers Launched: 2
27	[...] > Partial GroupAggregate (cost=18138.78..18139.64 rows=49 width=15) (actual time=687.189..693.467 rows=5379 loops=3)
28	[...] Group Key: l_1.nombre_producto
29	[...] > Sort (cost=18138.78..18138.90 rows=49 width=11) (actual time=687.149..688.450 rows=5398 loops=3)
30	[...] Sort Key: l_1.nombre_producto
31	[...] Sort Method: quicksort Memory: 418kB
32	[...] Worker 0: Sort Method: quicksort Memory: 502kB
33	[...] Worker 1: Sort Method: quicksort Memory: 419kB
34	[...] > Nested Loop (cost=0.42..18137.40 rows=49 width=11) (actual time=102.412..657.312 rows=5398 loops=3)
35	[...] > Nested Loop (cost=0.42..18133.14 rows=25 width=8) (actual time=102.309..500.633 rows=5328 loops=3)
36	[...] > Parallel Seq Scan on compra c_1 (cost=0.00..17872.74 rows=62 width=4) (actual time=102.005..361.674 rows=13192 loops=3)
37	[...] Filter: ((EXTRACT(year FROM fechaHora) = '2021'::numeric) AND (EXTRACT(month FROM fechaHora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
38	[...] Rows Removed by Filter: 320142
39	[...] > Index Only Scan using pk_delivery_correlativo on delivery d_1 (cost=0.42..4.20 rows=1 width=4) (actual time=0.009..0.009 rows=0 loops=39575)
40	[...] Index Cond: (correlativo = c_1.correlativo)

41	[...] Heap Fetches: 0
42	[...] -> Index Scan using idx_itemcompra on itemcompra i_1 (cost=0.0..0.15 rows=2 width=15) (actual time=0.021..0.027 rows=1 loops=15983)
43	[...] Index Cond: (correlativo = d_1.correlativo)
44	[...] Rows Removed by Index Recheck: 0
45	[...] -> Materialize (cost=19154.09..19154.12 rows=1 width=8) (actual time=0.031..0.032 rows=1 loops=16058)
46	[...] -> Aggregate (cost=19154.09..19154.10 rows=1 width=8) (actual time=504.298..504.403 rows=1 loops=1)
47	[...] -> Finalize GroupAggregate (cost=19138.80..19152.63 rows=117 width=15) (actual time=458.778..502.606 rows=16058 loops=1)
48	[...] Group Key: i_2.nombre_producto
49	[...] -> Gather Merge (cost=19138.80..19150.97 rows=98 width=15) (actual time=458.766..484.611 rows=16151 loops=1)
50	[...] Workers Planned: 2
51	[...] Workers Launched: 2
52	[...] -> Partial GroupAggregate (cost=18138.78..18139.64 rows=49 width=15) (actual time=394.875..398.738 rows=5384 loops=3)
53	[...] Group Key: i_2.nombre_producto
54	[...] -> Sort (cost=18138.78..18138.90 rows=49 width=11) (actual time=394.837..395.993 rows=5398 loops=3)
55	[...] Sort Key: i_2.nombre_producto
56	[...] Sort Method: quicksort Memory: 481kB
57	[...] Worker 0: Sort Method: quicksort Memory: 279kB
58	[...] Worker 1: Sort Method: quicksort Memory: 483kB
59	[...] -> Nested Loop (cost=0.42..18137.40 rows=49 width=11) (actual time=20.901..381.942 rows=5398 loops=3)
60	[...] -> Nested Loop (cost=0.42..18133.14 rows=25 width=8) (actual time=20.870..314.767 rows=5328 loops=3)
61	[...] -> Parallel Seq Scan on compra c_2 (cost=0.0..17872.74 rows=62 width=4) (actual time=20.735..240.512 rows=13192 loops=3)
62	[...] Filter: ((EXTRACT(year FROM fechaHora) = '2021'::numeric) AND (EXTRACT(month FROM fechaHora) = ANY ('{1,2,3,4,5,6}'::numeric[])))
63	[...] Rows Removed by Filter: 320142
64	[...] -> Index Only Scan using pk_delivery_correlativo on delivery d_2 (cost=0.42..4.20 rows=1 width=4) (actual time=0.005..0.005 rows=0 loops=39575)
65	[...] Index Cond: (correlativo = c_2.correlativo)
66	[...] Heap Fetches: 0
67	[...] -> Index Scan using idx_itemcompra on itemcompra i_2 (cost=0.0..0.15 rows=2 width=15) (actual time=0.010..0.012 rows=1 loops=15983)
68	[...] Index Cond: (correlativo = d_2.correlativo)
69	[...] Rows Removed by Index Recheck: 0
70	[...] -> Index Only Scan using pk_item_compra_nombre_correlativo_medida on itemcompra i (cost=0.42..4.43 rows=2 width=11) (actual time=0.089..0.091 rows=5 loops=1)
71	[...] Index Cond: (nombre_producto = (i_1.nombre_producto)::text)
72	[...] Heap Fetches: 0
73	[...] -> Index Scan using idx_delivery on delivery d (cost=0.0..0.04 rows=1 width=22) (actual time=0.015..0.015 rows=1 loops=5)
74	[...] Index Cond: (correlativo = i.correlativo)
75	[...] -> Sort (cost=7.39..7.74 rows=140 width=9) (actual time=0.179..0.197 rows=122 loops=1)
76	[...] Sort Key: r.dni
77	[...] Sort Method: quicksort Memory: 31kB
78	[...] -> Seq Scan on repartidor r (cost=0.0..2.40 rows=140 width=9) (actual time=0.013..0.030 rows=140 loops=1)
79	[...] -> Index Scan using idx_persona on persona p2 (cost=0.0..0.12 rows=1 width=15) (actual time=0.034..0.034 rows=1 loops=4)
80	[...] Index Cond: ((dni)::text = (d.rdn)::text)

81	[...] > Hash (cost=67266.94..67266.94 rows=179955 width=9) (actual time=3112.672..3112.677 rows=478194 loops=1)
82	[...] Buckets: 131072 (originally 131072) Batches: 8 (originally 4) Memory Usage: 3388kB
83	[...] > Hash Join (cost=47516.73..67266.94 rows=179955 width=9) (actual time=1936.149..2892.350 rows=478194 loops=1)
84	[...] Hash Cond: (cl.correlativo = b.correlativo)
85	[...] > Seq Scan on compralocal cl (cost=0.00..9966.48 rows=599248 width=4) (actual time=0.044..110.272 rows=600000 loops=1)
86	[...] > Hash (cost=44212.29..44212.29 rows=179955 width=17) (actual time=1935.606..1935.610 rows=478194 loops=1)
87	[...] Buckets: 65536 (originally 65536) Batches: 8 (originally 4) Memory Usage: 3585kB
88	[...] > Hash Join (cost=27903.49..44212.29 rows=179955 width=17) (actual time=634.701..1721.448 rows=478194 loops=1)
89	[...] Hash Cond: (b.correlativo = c.correlativo)
90	[...] > Seq Scan on boleta b (cost=0.00..8317.65 rows=539865 width=13) (actual time=0.019..93.399 rows=540000 loops=1)
91	[...] > Hash (cost=22445.99..22445.99 rows=332600 width=4) (actual time=633.986..633.987 rows=885256 loops=1)
92	[...] Buckets: 131072 (originally 131072) Batches: 16 (originally 8) Memory Usage: 3073kB
93	[...] > Seq Scan on compra c (cost=0.00..22445.99 rows=332600 width=4) (actual time=0.029..389.492 rows=885256 loops=1)
94	[...] Filter: (EXTRACT(year FROM fecha_hora) < '2021'::numeric)
95	[...] Rows Removed by Filter: 114744
96	Planning Time: 8.741 ms
97	JIT:
98	[...] Functions: 165
99	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true
100	[...] Timing: Generation 44.234 ms, Inlining 0.000 ms, Optimization 13.425 ms, Emission 348.007 ms, Total 405.667 ms
101	Execution Time: 4663.963 ms

Figura 64: Query Plan de la consulta 3 de 1m datos con índices

5.3.3.9. Explicación de la consulta

Sin índices: En los planes de consulta para las tablas 1k, 10k, 100k y 1m sin índices podemos hallar que se utilizan las primary key que poseen índice btree por defecto si es que los atributos a analizar lo poseen. Una gran cantidad de seqscan y nested loop inner join son encontrados en los planes de consulta y estos son ineficientes, debido a que los atributos no poseen índices adecuados, lo que genera un alto costo en la consulta.

Con índices: Para los planes de consulta con índices, se crearon tres índices del tipo hash: idx_itemcompra, idx_compra e idx_delivery. Estos índices reducen considerablemente, ya que optimiza el rendimiento al realizar los nested loop inner join.

5.4. Plataforma de pruebas

Sistema Operativo	Windows 10 64-bits
RAM	16 GB
CPU	Intel Core i7-10750H
Capacidad SSD	512 GB
Capacidad HDD	1 TB
PostgreSQL	14.0

Figura 65: Especificaciones plataforma de pruebas

5.5. Medición de tiempos

5.5.1. Sin índices

Consulta 1				
Ejecución	1k	10k	100k	1m
1	3.543	79.016	655.273	3242.406
2	3.968	42.911	495.606	1742.4
3	4.498	34.438	435.457	1753.317
4	4.151	31.497	433.781	1580.661
5	3.927	60.971	452.772	1421.85
Promedio	4.0174	49.7666	494.5778	1948.1268
Desviación Estándar	0.348027729	19.98004938	93.22452641	736.1099453

Figura 66: Consulta 1 sin índices

Consulta 2				
Ejecución	1k	10k	100k	1m
1	3.298	33.924	214.066	914.544
2	2.874	25.005	250.716	893.461
3	5.514	30.928	229.433	1062.255
4	3.706	27.944	220.974	900.889
5	4.646	28.429	298.454	1159.418
Promedio	4.0076	29.246	242.7286	986.1134
Desviación Estándar	1.067279	3.356396	34.06134	119.1657

Figura 67: Consulta 2 sin índices

Consulta 3				
Ejecución	1k	10k	100k	1m
1	6.579	244.155	857.432	4732.474
2	4.153	260.325	743.965	6084.732
3	7.012	287.040	778.950	5104.756
4	3.702	292.164	462.158	5194.353
5	5.000	266.538	780.534	5376.679
Promedio	5.289	270.044	724.608	5298.599
Desviación Estándar	1.306	17.636	136.366	445.655

Figura 68: Consulta 3 sin índices

5.5.2. Con índices

Consulta 1				
Ejecución	1k	10k	100k	1m
1	2.829	56.731	497.756	2241.368
2	3.137	46.183	414.292	2055.219
3	4.118	39.623	447.534	2067.558
4	3.368	69.435	413.528	2170.419
5	4.69	35.422	514.962	2275.146
Promedio	3.6284	49.4788	457.6144	2161.942
Desviación Estándar	0.760760343	13.75170027	46.96315244	99.36502504

Figura 69: Consulta 1 con índices

Consulta 2				
Ejecución	1k	10k	100k	1m
1	2.053	24.893	162.795	806.707
2	1.826	20.801	176.216	763.989
3	3.04	23.765	155.911	690.834
4	3.103	26.937	173.521	696.845
5	1.757	27.491	164.284	716.918
Promedio	2.3558	24.7774	166.5454	735.0586
Desviación Estándar	0.66283	2.686893	8.283257	49.28397

Figura 70: Consulta 2 con índices

Consulta 3				
Ejecución	1k	10k	100k	1m
1	3.528	234.198	543.809	4663.963
2	3.194	172.890	476.944	5457.421
3	3.195	163.751	469.753	3877.683
4	3.310	197.843	536.277	4238.072
5	3.503	228.703	691.358	4795.783
Promedio	3.346	199.477	543.628	4606.5844
Desviación Estándar	0.145	28.445	79.728	534.810

Figura 71: Consulta 3 con índices

5.6. Resultados

5.6.1. Consulta 1

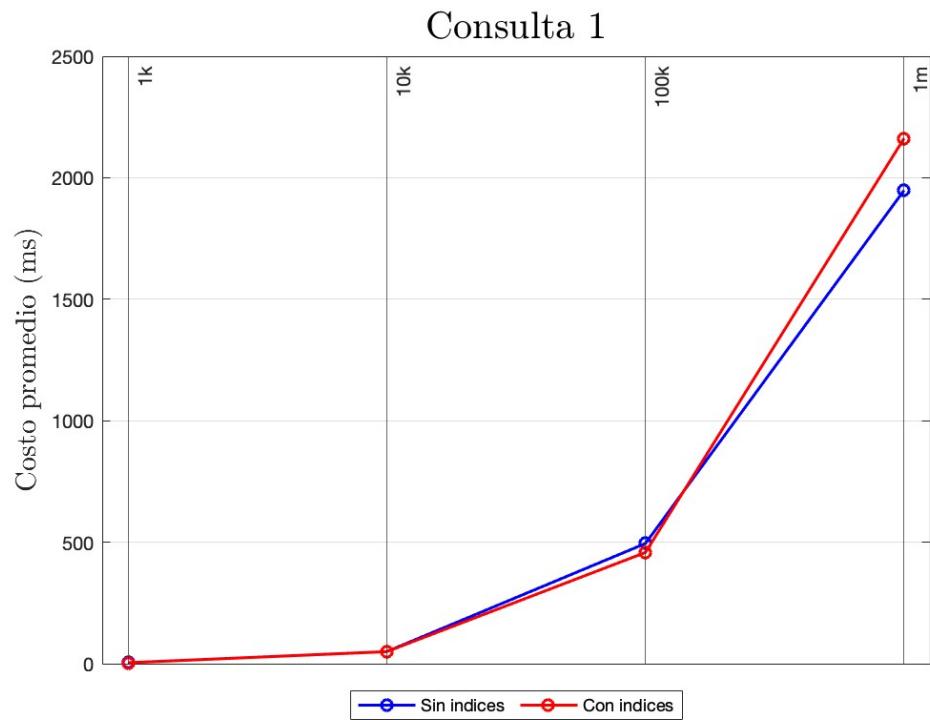


Figura 72: Gráfico consulta 1

5.6.2. Consulta 2

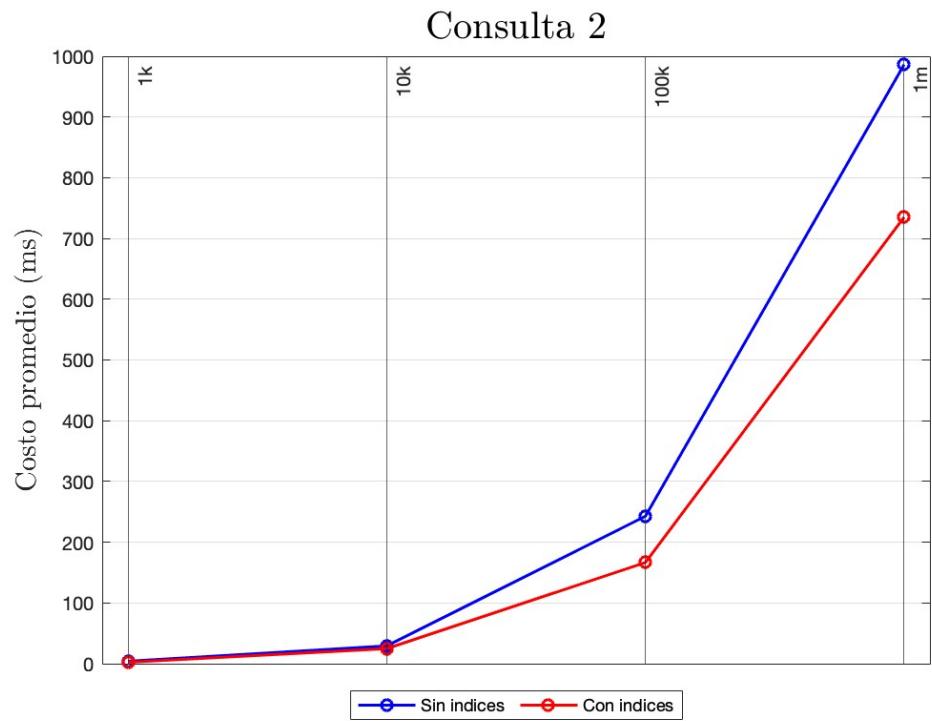


Figura 73: Gráfico consulta 2

5.6.3. Consulta 3

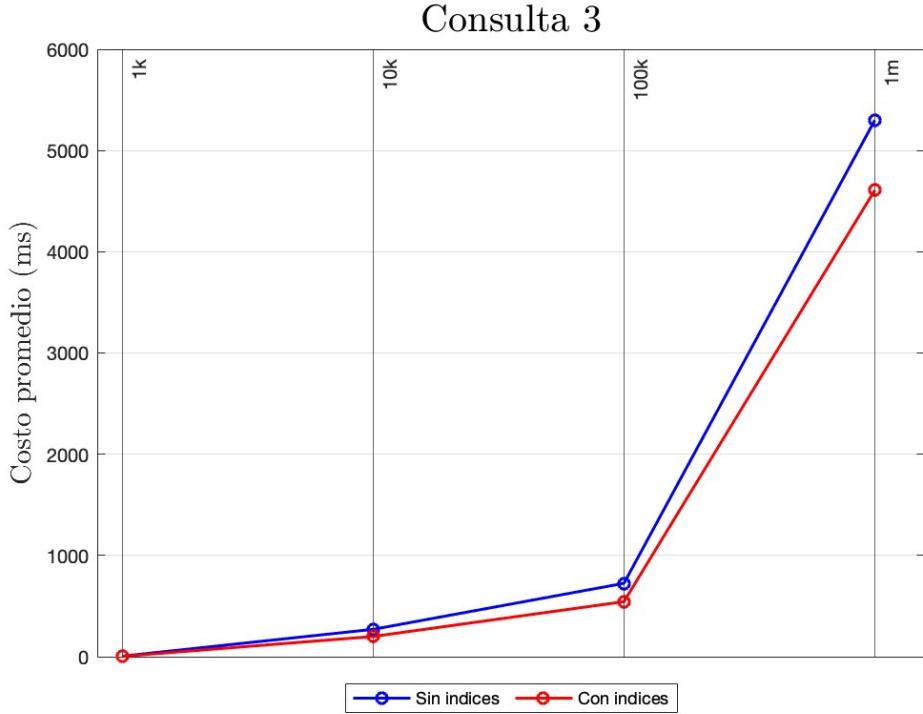


Figura 74: Gráfico consulta 3

5.7. Análisis y discusión

Luego de realizar la experimentación, se pudo observar notables mejoras en el rendimiento en las consultas 2 y 3 con índices. Sin embargo, en la consulta 1 se observó algo inesperado, pues el rendimiento de la consulta con índices apenas mejoró para los esquemas 1k, 10k y 100k, y empeoró para el esquema 1M. Creemos que esto se debe a que el índice se utilizó, no para ahorrar tiempo, sino para ahorrar memoria, puesto que en lugar de hacer un Hash-Join costoso en términos de memoria, utiliza un Nested-Loop con índices para hacer el Join de las tablas CompraLocal, Caja y Compra.

6. Conclusiones

1. El desarrollo del modelo entidad-relación y modelo relacional nos permitió implementar una base de datos consistente que representa el negocio de

la bodega Jessica cumpliendo lo requerido por la misma.

2. Se normalizó el modelo planteado y se implementó en SQL teniendo en cuenta las restricciones que impone la lógica de negocios, lo que nos llevó a implementar triggers y stored procedures para mantener la integridad de los datos.
3. El alto nivel de complejidad de las consultas sirvió para poder analizar los planes de consulta y entender la importancia de utilizar índices para optimizar las consultas, ya que la cantidad de tuplas crecen y es necesario contar con un sistema rápido.

7. Anexos

7.1. Normalización de la base de datos

En esta sección se demostrará que nuestra base de datos satisface la tercera forma normal.

Persona(dni, nombre)

$$F = \{dni \rightarrow nombre\}$$

dni es superllave de la relación. Por lo tanto, la relación está en la tercera forma normal.

Trabajador(Persona.dni, sueldo, celular)

$$F = \{dni \rightarrow sueldo, celular\}$$

dni es superllave. Está en la tercera forma normal.

ClienteDelivery(Persona.dni, celular)

$$F = \{dni \rightarrow celular\}$$

Dado que dni es superllave de la relación, satisface la tercera forma normal.

ClienteLocal(Persona.dni)

El conjunto de dependencias funcionales es trivial, por lo tanto, esta relación cumple con la tercera forma normal.

Turno(Trabajador.dni, día, hora_entrada, hora_salida)

$$F = \{dni, dia \rightarrow hora_entrada, hora_salida\}$$

(dni, dia) es superllave en el conjunto de dependencias funcionales dado. Por lo tanto, la relación está en la tercera forma normal.

Repartidor(Trabajador.dni, Local.direccion)

$$\mathbf{F} = \{dni \rightarrow direccion\}$$

dni es superllave en el conjunto de dependencias funcionales. Por ende, la relación satisface la tercera forma normal.

TrabajadorLocal(Trabajador.dni, Local.direccion, rol)

$$\mathbf{F} = \{dni \rightarrow direccion, rol\}$$

A simple vista dni es superllave. Existe una sola dependencia en donde el lado izquierdo cumple la condición. Está en la tercera forma normal.

Local(direccion, aforo, nombre, telefono)

$$\mathbf{F} = \{direccion \rightarrow aforo, nombre, telefono\}$$

Dirección es superllave dado que con esta podemos obtener toda la información de la entidad. En la única dependencia existente, se cumple la condición de la tercera forma normal.

Caja(Local.direccion, numero)

$$\mathbf{F} = \{direccion, numero\}$$

La presente dependencia funcional no presenta dependencias funcionales no triviales. Entonces se procede a juntar a ambas como superllave y al existir solo funciones triviales, la presente dependencia está en la tercera forma normal.

Producto(nombre_producto, medida, precio_unitario)

$$\mathbf{F} = \{nombre_producto, medida \rightarrow precio_unitario\}$$

(nombre_producto, medida) es superllave en el conjunto de dependencias funcionales. Por ende, la relación satisface la tercera forma normal

Empresa(ruc, razon_social)

El conjunto de dependencias funcionales es trivial, por lo tanto, esta relación cumple con la tercera forma normal.

Compra(correlativo, fecha_hora, metodo_pago, total)

$$\mathbf{F} = \{correlativo \rightarrow fecha_hora, metodo_pago, total\}$$

correlativo es superllave en el conjunto de dependencias funcionales. Por ende, la relación satisface la tercera forma normal

Delivery(Compra.correlativo, Repartidor.dni, ClienteDelivery.dni, direccion, costo_envio, fecha_hora_delivery)

$$F = \{correlativo \rightarrow dni, dni, direccion, costo_envio, fecha_hora_delivery\}$$

correlativo es superllave en el conjunto de dependencias funcionales. Por ende, se satisface la tercera forma normal.

CompraLocal(Compra.correlativo, Caja.numero, Caja.direccion)

$$F = \{correlativo \rightarrow numero, direccion\}$$

correlativo es superllave en el conjunto de dependencias funcionales. Por lo tanto, se satisface la tercera forma normal.

ItemCompra(Producto.nombre_producto, Producto.medida, Compra.correlativo, cantidad, subtotal)

$$F = \{nombre_producto, medida, correlativo \rightarrow cantidad, subtotal\}$$

El nombre del producto, su medida (unidades, kg, mg, etc) y el correlativo de la compra es la super llave del conjunto de dependencias funcionales, cumpliéndose la tercera forma normal.

Stock(Producto.nombre_producto, Producto.medida Local.direccion, cantidad)

$$F = \{nombre_producto, medida, direccion \rightarrow cantidad\}$$

El nombre del producto, su medida (unidades, kg, mg, etc) y la dirección del local es la superllave del conjunto de dependencias funcionales. Se satisface la tercera forma normal

EntregaProductos(Producto.nombre_producto,Producto.medida, Empresa.ruc, Local.direccion, fecha, cantidad)

$$F = \{nombre_producto, medida, ruc, direccion, fecha \rightarrow cantidad\}$$

El nombre del producto y la forma en que este se mide (unidades, kg, mg, etc), el ruc de la empresa proveedora, la dirección del local y la fecha de entrega

es la superllave del conjunto de dependencias funcionales. Por ello, se satisface la tercera forma normal.

Boleta(CompraLocal.correlativo, ClienteLocal.dni)

$$F = \{correlativo \rightarrow numero, dni\}$$

correlativo es superllave en el conjunto de dependencias funcionales. Por lo tanto, se cumple la tercera forma normal.

Factura(CompraLocal.correlativo, Empresa.ruc)

$$F = \{correlativo \rightarrow ruc\}$$

correlativo es superllave en el conjunto de dependencias funcionales. Por lo tanto, se cumple la tercera forma normal.

EncargadoCaja(TrabajadorLocal.dni, Caja.numero, Caja.direccion)

$$F = \{dni \rightarrow numero, direccion\}$$

dni es superllave, entonces, se satisface la tercera forma normal.

7.2. Dump Data

Código de generación de cada tabla por esquema: <https://drive.google.com/drive/folders/1CdsQU6liRNQ-zTDkbyfpaoJ7LEGviFMl?usp=sharing>

7.3. Modelo Físico

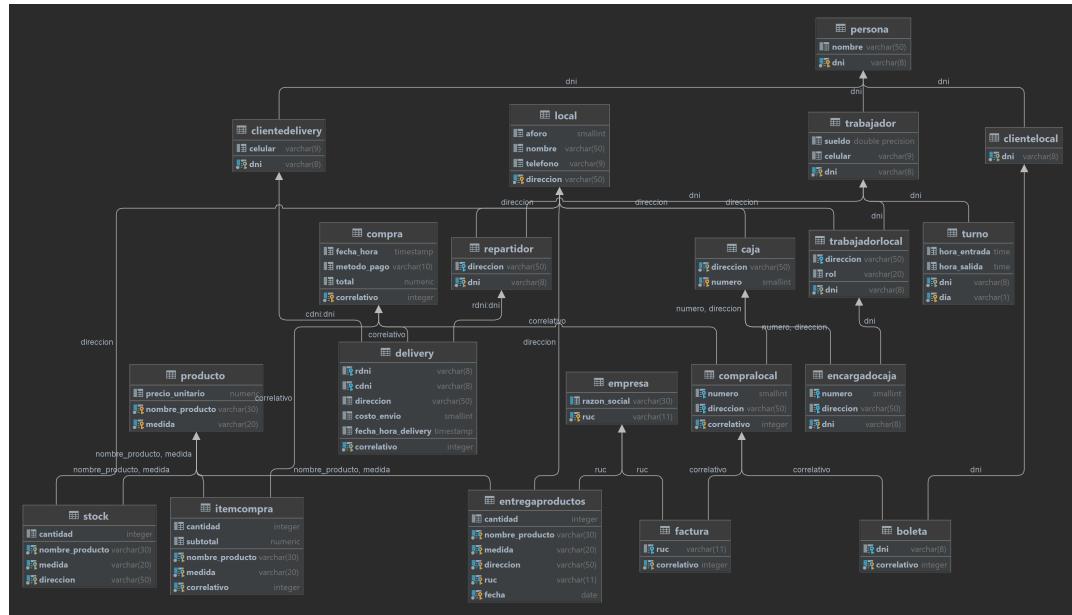


Figura 75: Modelo físmico

7.4. Videos de experimentación

7.4.1. Consulta 1

https://drive.google.com/file/d/1dBk4C0rU_dyzt8aXnby1jk05Du9gf0YE/view?usp=sharing

7.4.2. Consulta 2

<https://drive.google.com/file/d/11rZFZxFcGlFSqedv0Yz30ueAMx9masw3/view?usp=sharing>

7.4.3. Consulta 3

https://drive.google.com/file/d/1qm_SDgZ44nrcbhPfSBSGccWMg10hRyz8/view?usp=sharing

7.5. Frontend para la base de datos

<https://github.com/jjordanoc/proyectobd>

7.6. Pregunta extra

¿Cuál sería la complejidad operacional si escalamos los datos por encima del millón?, realice una comparativa respecto a la cantidad de datos del párrafo anterior. ¿Es suficiente la arquitectura Cliente-Servidor para procesar millones de datos?

Con datos por encima del millón, la complejidad operacional sería considerablemente mayor. Posiblemente, esto nos obligaría a crear más índices para optimizar las consultas más frecuentes de tal forma que estas sean realizables en un tiempo razonable. Asimismo, la arquitectura cliente-servidor sí es suficiente para procesar esta cantidad de datos, pues gracias a la separación de la interfaz con la que interactúa el usuario de la lógica del negocio, se pueden manejar con facilidad las operaciones de manera asíncrona.