



CS-2041 Base de Datos 1 - Grupo 1

Hito 1 del proyecto

Ciclo 2024-1

Creación de una base de datos para el policlínico MEDICENTER SAC

Elaborado por

Michael Hinojosa - 202310528 (100 %)

Federico Iribar - 202310321 (50 %)

Daniel Caballero - 202310413 (100 %)

Profesor:

Mg. Teófilo Chambilla

tchambilla@utec.edu.pe

Asistentes de cátedra:

Samantha Chang

Valeria Espinoza

Trabajo presentado:

17.05.2024

Índice

1. Requisitos	1
1.1. Introducción	1
1.2. Descripción del problema/organización/empresa	1
1.3. Necesidad/usos de la base de datos	1
1.4. ¿Cómo resuelve el problema hoy?	2
1.4.1. ¿Cómo se almacenan/procesa los datos hoy?	2
1.4.2. Flujo de datos	2
1.5. Descripción detallada del sistema	2
1.5.1. Objetos de información actuales	2
1.5.2. Características y funcionalidades esperadas	2
1.5.3. Tipos de usuarios existentes/necesarios	3
1.5.4. Tipos de consulta, actualizaciones	3
1.5.5. Tamaño estimado de la base de datos	4
1.6. Objetivos del proyecto	6
1.7. Referencias del proyecto	6
1.8. Eventualidades	6
1.8.1. Problemas que pudieran encontrarse en el proyecto	6
1.8.2. Límites y alcances del proyecto	6
2. Modelo Entidad-Relación	7
2.1. Reglas semánticas	7
2.2. Modelo Entidad-Relación	8
2.3. Especificaciones y consideraciones sobre el modelo	8
2.3.1. Entidad Persona:	8
2.3.2. Entidad Secretaria:	8
2.3.3. Entidad Doctor:	8
2.3.4. Entidad Interno:	9
2.3.5. Entidad Paciente:	9
2.3.6. Entidad Enfermera:	9
2.3.7. Entidad Gerente:	9
2.3.8. Entidad Pedido:	9
2.3.9. Entidad Insumo médico:	9
2.3.10. Entidad Actividad Económica:	10
2.3.11. Entidad Operación:	10
2.3.12. Entidad Consulta:	10
2.3.13. Entidad Pago:	10
3. Modelo Relacional	11
3.1. Modelo Relacional	11
3.2. Especificaciones de transformación	11

3.2.1.	Entidades	11
3.2.2.	Entidades débiles	12
3.2.3.	Entidades superclase/subclases	12
3.2.4.	Relaciones binarias	13
3.2.5.	Relaciones ternarias	13
3.3.	Diccionario de datos	14
3.3.1.	Tabla de Persona	14
3.3.2.	Tabla de Gerente	14
3.3.3.	Tabla de Enfermera	14
3.3.4.	Tabla de Interno	14
3.3.5.	Tabla de Doctor	15
3.3.6.	Tabla de Paciente	15
3.3.7.	Tabla de Secretaria	15
3.3.8.	Tabla de Pedido	15
3.3.9.	Tabla de Insumo médico	16
3.3.10.	Tabla de detalle Pedido	16
3.3.11.	Tabla de Actividad Económica	16
3.3.12.	Tabla de Consulta	17
3.3.13.	Tabla de Operación	17
3.3.14.	Tabla de Reserva	17
3.3.15.	Tabla de Utiliza	18
3.3.16.	Tabla de Pago	18
4.	Implementación de la base de datos	19
4.1.	Creación de Tablas en PostgreSQL	19
4.2.	Carga de datos	25
5.	Optimización y Experimentación	26
5.1.	Consultas SQL para el Experimento	26
5.1.1.	Descripción del Tipo de Consultas Seleccionadas	26
5.1.2.	Implementación de Consultas en SQL	27
5.2.	Metodología del experimento	28
5.2.1.	Experimentación sin índices	28
5.2.2.	Experimentación con índices	28
5.3.	Optimización de consultas	29
5.3.1.	Planes de índices para Consulta 1	29
5.3.2.	Planes de índices para Consulta 2	30
5.3.3.	Planes de índices para Consulta 3	30
5.4.	Plataforma de Pruebas	30
5.5.	Medición de tiempos	31
5.5.1.	Ejecucion de consulta 1	31
5.5.2.	Ejecución de consulta 2	40

5.5.3. Ejecucion de consulta 3	52
5.5.4. Sin índices	59
5.5.5. Con índices	60
5.6. Resultados	61
5.6.1. Consulta 1	61
5.6.2. Consulta 2	62
5.6.3. Consulta 3	62
5.7. Análisis y Discusión	63
5.7.1. Comparación índices vs no índices	63
6. Conclusiones	63
7. Anexo	64
7.1. Link al Drawio del Modelo de Entidad Relación	64
7.2. Github	64
7.3. Data <i>Dump</i>	64
7.4. Script para poblar base de datos	64

Índice de figuras

1. Query Plan para la consulta 1 con índices en la tabla de 1 mil	31
2. Query Plan para la consulta 1 sin índices en la tabla de 1 mil	32
3. Query Plan Tree para la planificación con índices en la tabla de 1 mil	32
4. Query Plan Tree para la planificación sin índices en la tabla de 1 mil	33
5. Query Plan para la consulta 1 con índices en la tabla de 10 mil	33
6. Query Plan para la consulta 1 sin índices en la tabla de 10 mil	34
7. Query Plan Tree para la planificación con índices en la tabla de 10 mil	34
8. Query Plan Tree para la planificación sin índices en la tabla de 10 mil	34
9. Query Plan para la consulta 1 con índices en la tabla de 100 mil (primera parte)	35
10. Query Plan para la consulta 1 sin índices en la tabla de 100 mil	36
11. Query Plan Tree para la planificación con índices en la tabla de 100 mil	36
12. Query Plan Tree para la planificación sin índices en la tabla de 100 mil	37
13. Query Plan para la consulta 1 con índices en la tabla de 1 millón	37
14. Query Plan para la consulta 1 sin índices en la tabla de 1 millón	38
15. Query Plan Tree para la planificación con índices en la tabla de 1 millón	38
16. Query Plan Tree para la planificación sin índices en la tabla de 1 millón	39
17. Query Plan para la consulta 2 con índices en la tabla de 1 mil	40
18. Query Plan para la consulta 2 sin índices en la tabla de 1 mil	41
19. Query Plan Tree para la planificación con índices en la tabla de 1 mil	42
20. Query Plan Tree para la planificación sin índices en la tabla de 1 mil	42
21. Query Plan para la consulta 2 con índices en la tabla de 10 mil	43

22.	Query Plan para la consulta 2 sin índices en la tabla de 10 mil	44
23.	Query Plan Tree para la planificación con índices en la tabla de 10 mil	45
24.	Query Plan Tree para la planificación sin índices en la tabla de 10 mil	45
25.	Query Plan para la consulta 2 con índices en la tabla de 100 mil	46
26.	Query Plan para la consulta 2 sin índices en la tabla de 100 mil	47
27.	Query Plan Tree para la planificación con índices en la tabla de 100 mil	48
28.	Query Plan Tree para la planificación sin índices en la tabla de 100 mil	48
29.	Query Plan para la consulta 2 con índices en la tabla de 1 millón	49
30.	Query Plan para la consulta 2 sin índices en la tabla de 1 millón	50
31.	Query Plan Tree para la planificación con índices en la tabla de 1 millón	51
32.	Query Plan Tree para la planificación sin índices en la tabla de 1 millón	51
33.	Query Plan para la consulta 3 con índices en la tabla de 10 mil	52
34.	Query Plan Tree para la consulta 3 con índices en la tabla de 10 mil	52
35.	Query Plan para la consulta 3 sin índices en la tabla de 10 mil	53
36.	Query Plan Tree para la consulta 3 sin índices en la tabla de 10 mil	53
37.	Query Plan Tree para la consulta 3 con índices en la tabla de 100 mil	54
38.	Query Plan para la consulta 3 con índices en la tabla de 100 mil	55
39.	Query Plan para la consulta 3 sin índices en la tabla de 100 mil	56
40.	Query Plan Tree para la consulta 3 sin índices en la tabla de 100 mil	56
41.	Query Plan Tree para la consulta 3 con índices en la tabla de 1 millón	57
42.	Query Plan para la consulta 3 con índices en la tabla de 1 millón	58
43.	Query Plan para la consulta 3 sin índices en la tabla de 1 millón	59
44.	Query Plan para la consulta 3 sin índices en la tabla de 1 millón	59
45.	Comparación de la consultan 1	61
46.	Comparación de la consulta 2	62
47.	Comparación de la consulta 3	62

Índice de cuadros

1.	Estimación de la Base de Datos	4
2.	Tamaño de las tablas fijas	5
3.	Tamaño de las tablas cambiantes	5
4.	Tabla de Persona	14
5.	Tabla de Gerente	14
6.	Tabla de Enfermera	14
7.	Tabla de Interno	14
8.	Tabla de Doctor	15
9.	Tabla de Paciente	15
10.	Tabla de Secretaria	15
11.	Tabla de Pedido	15
12.	Tabla de Insumo médico	16
13.	Tabla de detalle Pedido	16

14.	Tabla de Actividad Económica	17
15.	Tabla de Consulta	17
16.	Tabla de Operación	17
17.	Tabla de Reserva	18
18.	Tabla de Utiliza	18
19.	Tabla de Pago	18

1. Requisitos

1.1. Introducción

La empresa Medicenter SAC es un policlínico que fue fundado por un doctor en la década de los 90s. En aquellos tiempos la empresa comenzó como el consultorio del doctor Daniel Alcides Carrión, quien después de estar disconforme con cómo trataban a sus pacientes en el hospital donde trabajaba, decidió independizarse.

Al inicio este pequeño consultorio recibía pocas personas al día, por lo que el manejo de datos era sencillo y solo se usaban unas cuántas tablas en hojas cuadriculadas, creadas por el mismo doctor en las tardes cuando no tenía pacientes. Sin embargo con el paso de los años, la buena atención del doctor Carrión hizo que ganara una buena reputación como doctor y más personas lo buscaran para ser atendidas.

Luego de unos años, el doctor Carrión logró juntar suficiente dinero para ampliar su consultorio y poder ofrecer operaciones ambulatorias de corta duración. La ampliación de las operaciones hizo que el doctor Carrión necesite ayuda y como consiguiente contrató a una secretaria y una enfermera que lo ayudara en las operaciones ambulatorias. Al estar ocupado con la lógica del nuevo negocio, dejó de lado la importancia de manejar bien los datos y siguió guardando toda la información en cuadernos.

Con el pasar de los años, el doctor Carrión tuvo hijos que luego crecieron. Uno de sus hijos estudió Ingeniería Industrial y decidió ayudar a su padre con la rama administrativa de la empresa. Durante este tiempo el policlínico había quintuplicado sus operaciones, sin embargo seguían guardando todos los datos en cuadernos o de vez en cuando en hojas de cálculo de excel.

1.2. Descripción del problema/organización/empresa

Tras hacer un análisis detallado de la empresa, el hijo determinó que lo primordial para mantener un crecimiento ordenado era tener una base de datos personalizada a sus necesidades para poder migrar los datos de los cuadernos y hojas de excel que tenían hacia su base de datos.

1.3. Necesidad/usos de la base de datos

La empresa busca modernizarse para mantener su crecimiento acelerado y para ello requiere de una base de datos que le pueda brindar un crecimiento escalable. Lo más importante para la empresa es no perder información, ya que no hay nada peor que no poder hacer el seguimiento de un paciente porque se traspapeló su historia clínica.

1.4. ¿Cómo resuelve el problema hoy?

1.4.1. ¿Cómo se almacenan/procesa los datos hoy?

Actualmente los datos se almacenan en cuadernos, hojas de cálculo e historias clínicas físicas. Todas las historias clínicas son guardadas al final de la jornada laboral en un Pioner, en donde se guardan todas las historias clínicas de aquel mes.

Las facturas y boletas son recolectadas y al final de cada semana son pegadas en hojas A4, para que al final del mes el contador pueda revisarlas y hacer los balances de cuentas apropiados.

1.4.2. Flujo de datos

Actualmente el principal flujo de datos se da con los pacientes. Cada paciente tiene una historia clínica que debe ser almacenada en la base de datos. Cada vez que el paciente tiene una consulta con el doctor, esta historia clínica debe ser actualizada con la información adicional, para así tener un historial de la persona.

Asimismo, cada vez que el paciente realiza una consulta con el doctor debe de pagar en efectivo o como transferencia bancaria y esa información debe de ser recopilada

Además, cada vez que hay una operación, se utilizan insumos como guantes de látex, y demás material médico del que se tiene que mantener un histórico de cuánto queda, para comprar más cuando queda poco.

1.5. Descripción detallada del sistema

1.5.1. Objetos de información actuales

La información actual se encuentra guardada en decenas de pioners en un cuarto del local en donde el doctor Carrión atiende a sus pacientes. Asimismo, desde que se contrató a un nuevo contador, este ha comenzado a trabajar con excel, sin embargo ello significa un porcentaje pequeño de datos en comparación con lo que hay. Adicional a los datos ya existentes, se realizarán scripts para generar nuevos datos, y así poner a prueba el sistema para consultas que involucran 1.000 (mil) datos, 10.000 (diez mil) datos, 100 000 (cien mil) datos y 1 000 000 (un millón on) de datos

1.5.2. Características y funcionalidades esperadas

El policlínico espera tener una base de datos robusta y escalable para tener un mejor seguimiento de sus pacientes y para ello se requiere una base de datos que pueda manejar

un gran nivel de datos.

=Asimismo, al poder tener todos los datos en una misma base de datos, la empresa busca poder hacer mejores análisis de costos y productividad, ya que tendrán acceso a todos los datos al mismo tiempo.

Por último la empresa está interesada en poder tener un mejor registro del inventario de insumos e instrumental médico que tienen, para saber con mayor facilidad que insumos se debe comprar y cuales no.

1.5.3. Tipos de usuarios existentes/necesarios

El policlínico cuenta actualmente con 6 trabajadores. El dueño y doctor principal es el doctor Carrión, sin embargo el año pasado comenzó a contratar anualmente a 1 médico recién graduado de la carrera de medicina en calidad de "interno" para que lo ayude durante las consultas y operaciones. Asimismo se cuenta con 1 enfermera que ayuda durante las operaciones y hace los chequeos pre operacionales con el interno. Además, la empresa cuenta con una secretaria que reserva las consultas. Por último el cargo de Gerente lo ocupa el hijo del doctor Carrión. Por último se cuenta con un contador que lleva toda la parte contable de la empresa.

Es así como los usuarios identificados que accederán a la base de datos son:

- **Personal administrativo:**

El personal administrativo contará con el acceso a toda la base de datos y podrá divisar los datos de los pacientes, poder hacer consultas sobre los gastos e ingresos y poder revisar el inventario para determinar qué se debe comprar a futuro.

- **Secretaria:**

La secretaria contará con acceso a los datos de los pacientes para poder extraer e imprimir las historias clínicas de los pacientes que tendrán consulta con el doctor.

- **Contador:**

El contador contará con acceso a la parte contable de la base de datos y actualizará los pagos y facturas que se realicen.

- **Interno y Enfermera:** El Interno y la Enfermera contarán con acceso a la parte de inventario para poder sacar los insumos médicos que se utilizarán en la operación.

1.5.4. Tipos de consulta, actualizaciones

Algunas de las consultas que se podrán hacer en la base de datos:

- ¿Cuánto se gastó en insumos médicos en los últimos 30 días?

- ¿A cuánto ascienden los ingresos en el primer trimestre del año?
- ¿Cuántas consultas ha tenido el paciente "Juan Perez." en los últimos 3 años?
- ¿Cuál fue el mes con mayor operaciones en la historia del policlínico?
- Devolver el salario de todo el personal del policlínico
- ¿Cuál es el tipo de operación más común en el policlínico?
- ¿Cuál es el tipo de operación que más dinero genera luego de contabilizar los gastos en insumos médicos?

Algunas de las actualizaciones que se podrán hacer en la base de datos:

- Actualizar el sueldo del interno
- Borrar a los internos de medicina que acabaron su periodo de un año
- Actualizar el historial clínico de un paciente
- Actualizar la cantidad de algún insumo médico

1.5.5. Tamaño estimado de la base de datos

Cuadro 1: Estimación de la Base de Datos

Nombre de la Tabla	Longitud de los Atributos (Bytes)	Longitud de los Registros (Bytes)
Persona	8+25+25+8	66
Gerente	8+8	16
Pedido	10+8+8	26
Insumo medico	10+50+30+8+50+8	156
detalle Pedido	10+10+2	22
Enfermera	8+8	16
Interno	8+8	16
Doctor	8+50+8	66
Paciente	8	8
Secretaria	8+8	16
Actividad Económica	8+8+8+10+250+8	292
Consulta	10+8+8	26
Reserva	10+8+8	26
Operación	10+8+50	68
utiliza	10+10+4	24
Pago	10+10+10+8+8	46

Cuadro 2: Tamaño de las tablas fijas

Nombre de la Tabla	Tamaño en Bytes	Datos Estimado	Tamaño Total
Gerente	16	1	16
Insumo medico	150	4	600
Enfermera	16	1	16
Interno	16	1	16
Doctor	66	1	66
Secretaria	16	1	16
Total			730

Cuadro 3: Tamaño de las tablas cambiantes

Nombre de la Tabla	Tamaño en Bytes	Tamaño Estimado (M)	Tamaño Total
Persona	66	30	1 980
Pedido	26	30	780
detalle Pedido	22	30	660
Paciente	8	30	240
Actividad Económica	292	300	87 600
Consulta	26	40	1 040
Reserva	26	30	780
Operación	68	30	2 040
utiliza	24	40	960
Pago	46	300	13 800
Total			109 880

1.6. Objetivos del proyecto

Los principales objetivos de este proyecto son:

1. Aplicar lo aprendido en el curso y crear una base de datos robusta y escalable que respete las reglas semánticas proporcionadas por el cliente.
2. Implementar el modelo de base de datos en el DBMS PostgreSQL.
3. Crear consultas óptimas para no utilizar recursos innecesarios.

1.7. Referencias del proyecto

Para realizar el proyecto se utilizó el conocimiento general de policlínicos y hospitales de los integrantes del proyecto.

1.8. Eventualidades

1.8.1. Problemas que pudieran encontrarse en el proyecto

Un problema a considerar sería la cancelación de cirugías ya programadas debido a circunstancias de fuerza mayor. En ese caso se tendría que reprogramar la cirugía y en caso ya se hayan separado y seleccionados insumos médicos del inventario se tendrían que retornar

1.8.2. Límites y alcances del proyecto

Esta base de datos está modelada en base al policlínico Medicenter SAC y sus necesidades actuales. Actualmente Medicenter solo ofrece cirugías ambulatorias que no requieren hospitalización, por lo que en este modelo de base de datos no se ha tomado en cuenta la reserva de cuartos para la hospitalización.

Al igual, como solo hay una sala de operación y un solo doctor, no se ha modelado la sala de operación, puesto que no hay información necesaria. Lo mismo aplica para la oficina del doctor Alcides Carrion, en donde se realizan las consultas.

Asimismo para el modelado de esta base de datos se ha considerado que el único doctor es el doctor Alcides Carrion, en caso se expanda y se tenga múltiples consultas y cirugías al mismo tiempo se tendrá que crear entidades adicionales que satisfacen esos casos.

Por último, no se consideró el horario del doctor Alcides Carrion y se dejó a discreción del policlínico ese detalle.

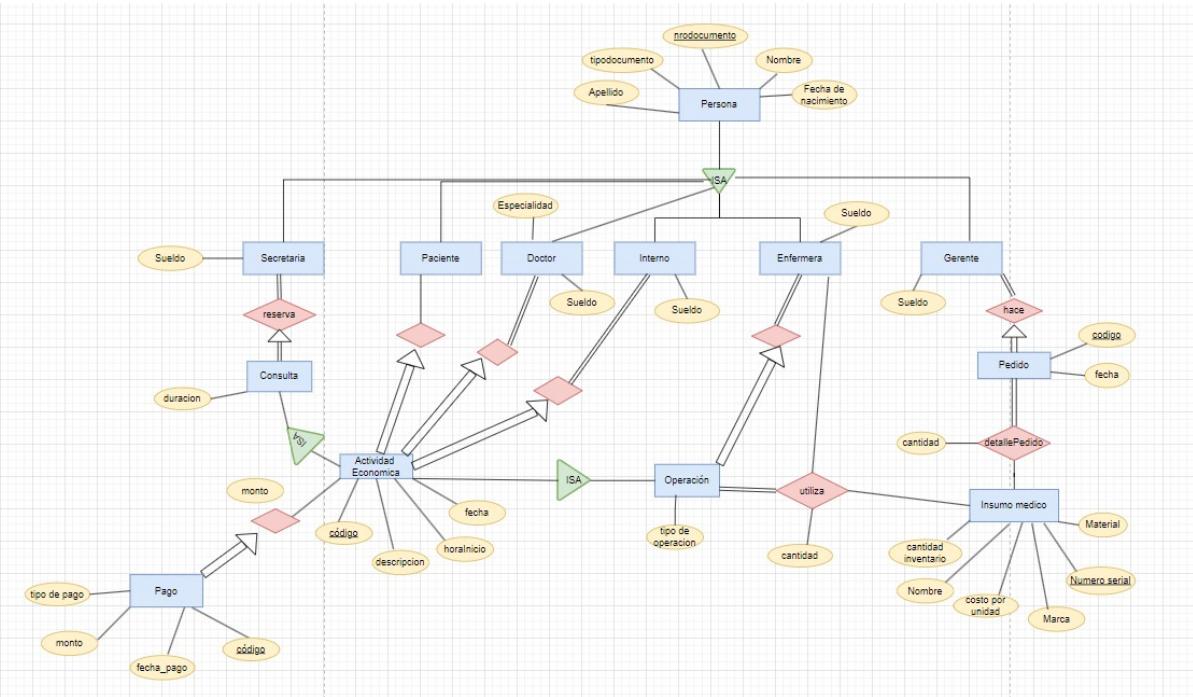
2. Modelo Entidad-Relación

2.1. Reglas semánticas

El modelado de la base de datos, comenzando por el modelo de Entidad-Relación, se hará tomando en cuenta las siguientes reglas semánticas.

1. El policlínico cuenta con una sala de operaciones ambulatoria y una oficina que sirve de consultorio para el doctor Alcides Carrion. Además, cuenta con un cuarto donde se hacen los chequeos pre operatorios y donde el paciente es llevado luego de la operación hasta que se despierte de la operación. Ningún paciente se queda a dormir en el policlínico.
2. Las principales actividades del doctor son operar y hacer consultas. El interno está presente ambas actividades. En las operaciones también está presente una enfermera que ayuda al doctor y al interno.
3. El doctor se identifica con su número de dni. También tiene nombre, apellido, fecha de nacimiento y especialidad. El interno se identifica por su dni y tiene nombre, apellido, fecha de nacimiento y aún no tiene especialidad. Por último las enfermeras se identifican por su dni y tienen fecha de nacimiento, año y nombre.
4. Durante las operaciones se utilizan insumos médicos. Los insumos médicos son retirados un día antes por la enfermera y/o los internos del inventario. Los insumos médicos se identifican por un número serial y tienen nombre, marca, el material del que están fabricados, la cantidad que hay en el inventario y un costo por unidad.
5. Las operaciones no tienen una duración exacta ya que pueden haber complicaciones. Las operaciones pueden ser de diferente tipo y se identifican por el código de la actividad económica.
6. Las consultas sí tienen una duración limitada de media hora. En las consultas participan el doctor, el interno de turno y el paciente. Las consultas son reservadas mediante la secretaria que es la que está en comunicación con los pacientes. Las consultas se identifican por el código de la actividad económica y su hora de inicio.
7. La secretaria se identifica con su número de dni y tiene nombre, apellido y edad.
8. Toda actividad económica tiene un pago y se identifica por un código y tiene fecha y la descripción de la actividad económica.
9. El gerente se encarga de la compra de insumos médicos. A inicios del mes, el gerente revisa la cantidad de inventario que hay y determina qué insumos médicos deben ser comprados.
10. Todos los pagos se identifican por un código y tienen fecha, monto, el tipo de pago y el código de la actividad con la que están relacionados.

2.2. 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), nombre, apellido y su fecha de nacimiento.
- Consideraciones:** La llave primaria de Persona es su DNI, ya que es la forma más sencilla de identificar a una persona. Esta entidad es una superclase, la cual hereda a las subclases Secretaria, Paciente, Doctor, Interno, Enfermera y Gerente.

2.3.2. Entidad Secretaria:

- Especificaciones:** La llave primaria de Secretaria es a su vez foránea, el DNI de la Entidad Persona.
- Consideraciones:** Esta entidad es una subclase que hereda los atributos de Persona.

2.3.3. Entidad Doctor:

- Especificaciones:** La llave primaria de Doctor es a su vez foránea, el DNI de la Entidad Persona.
- Consideraciones:** Esta entidad es una subclase que hereda los atributos de Persona.

2.3.4. Entidad Interno:

- **Especificaciones:** La llave primaria de Interno es a su vez foránea, el DNI de la Entidad Persona.
- **Consideraciones:** Esta entidad es una subclase que hereda los atributos de Persona.

2.3.5. Entidad Paciente:

- **Especificaciones:** La llave primaria de Doctor es a su vez foránea, el DNI de la Entidad Paciente.
- **Consideraciones:** Esta entidad es una subclase que hereda los atributos de Persona.

2.3.6. Entidad Enfermera:

- **Especificaciones:** La llave primaria de Doctor es a su vez foránea, el DNI de la Entidad Enfermera.
- **Consideraciones:** Esta entidad es una subclase que hereda los atributos de Persona.

2.3.7. Entidad Gerente:

- **Especificaciones:** La llave primaria de Doctor es a su vez foránea, el DNI de la Entidad Gerente.
- **Consideraciones:** Esta entidad es una subclase que hereda los atributos de Persona.

2.3.8. Entidad Pedido:

- **Especificaciones:** Almacena el código del pedido y la fecha en el que fue enviado; el código será la llave primaria de esta entidad.
- **Consideraciones:** El código funcionará fácil como llave de esta entidad debido a que se generará uno nuevo con cada pedido lo que hará fácil el hecho de que no se repitan.

2.3.9. Entidad Insumo médico:

- **Especificaciones:** Almacena el número serial del insumo, la marca de este, el material del que este hecho, su costo, el nombre de este, y la cantidad que haya en inventario; el número serial será la llave primaria de esta entidad.

- **Consideraciones:** El número serial funcionará fácil como llave de esta entidad debido a que se generará un nuevo número serial con cada insumo médico que sea fabricado lo que hará fácil el hecho de que no se repitan.

2.3.10. Entidad Actividad Económica:

- **Especificaciones:** Almacena la información más importante de cada Actividad económica: su código es compuesto y consta de su código y de la horaInicio. Tiene de atributos: la fecha en la que se realizó la actividad económica y la descripción de la actividad económica, es decir la historia clínica. Además cuenta con la duración de la actividad económica
- **Consideraciones:** La llave primaria de Actividad Económica es su código y la horaInicio. Se podría considerar a la fecha como una llave potencial, sin embargo puede haber más de una actividad económica por día y por ende no funcionaría.

2.3.11. Entidad Operación:

- **Especificaciones:** Almacena la información básica de la operación. La llave es compuesta y foránea del código y horainicio de la actividad económica.
- **Consideraciones:** Operación es una subclase de Actividad económica, por lo que no tiene llave primaria por su cuenta.

2.3.12. Entidad Consulta:

- **Especificaciones:** La llave de Consulta es compuesta y foránea. Consta del código de Actividad económica y Hora Inicio
- **Consideraciones:** Consulta es una subclase de Actividad económica, por lo que no tiene llave primaria por su cuenta.

2.3.13. Entidad Pago:

- **Especificaciones:** Esta entidad posee una llave primaria (Código de Pago). Además, incluye el monto, la fecha de pago y el tipo de pago.
- **Consideraciones:** Pago recibe una llave foránea de Actividad Económica (código), para poder relacionar el pago con la actividad económica con la que se relaciona.

3. Modelo Relacional

3.1. Modelo Relacional

- **Persona**(dni, nombre, apellido, fecha de nacimiento)
- **Gerente**(Persona.dni, sueldo)
- **Pedido**(código, Gerente.dni, fecha)
- **Insumo medico**(número serial, material, marca, costo, nombre, cantidad inventario)
- **detalle Pedido**(Pedido.código, Insumo medico.numero serial, cantidad)
- **Enfermera**(Persona.dni, sueldo)
- **Interno**(Persona.dni, sueldo)
- **Doctor**(Persona.dni, Especialidad, sueldo)
- **Paciente**(Persona.dni)
- **Secretaria**(Persona.dni, sueldo)
- **Actividad Económica**(Paciente.dni, Doctor.dni, Interno.dni, código, horaInicio descripción, fecha)
- **Consulta**(Actividad Económica.código, Secretaria.dni, duración)
- **Operación**(Actividad Económica.código, Enfermera.dni, tipo de operación)
- **utiliza**(Operación.código, Insumo medico.numero serial, Enfermera.dni, cantidad)
- **Pago**(código, Actividad Económica.código, tipo de pago, monto, fecha)

3.2. Especificaciones de transformación

Para la transformación de las entidades se tomó en cuenta lo siguiente:

3.2.1. Entidades

- **Pago:** Su llave primaria es su código y tiene como atributos el tipo de pago que se realizó (yape, transferencia ...), el monto y la fecha en el que se realizó el pago.
- **Insumo médico:** Su llave primaria es el número serial y tiene de atributos el material del que está hecho, la cantidad en inventario que hay de este insumo, el nombre, el costo por unidad y su marca.
- **Pedido:** Su llave primaria es su código y tiene de atributo la fecha en el que se hizo el pedido. Además recibe como llave foránea el dni de Gerente.

3.2.2. Entidades débiles

No se modeló entidades débiles para esta base de datos

3.2.3. Entidades superclase/subclases

- **Superclase (Persona):** Esta superclase tiene como llave primaria el dni y cuenta con los atributos de nombre, edad y apellido. Esta subclase hereda hacia 6 entidades y hay solapamiento por lo que se crea la tabla Persona. Por ejemplo, puede ser que el paciente sea el gerente o la enfermera.

- **Subclases**

- **Secretaria** La secretaria puede también ser paciente en algún momento, lo que conlleva a que haya solapamiento. La llave de la secretaría es el dni, que recibe como llave foránea de persona y tiene de atributo su sueldo.
 - **Paciente** El paciente puede en algún momento trabajar en la empresa por lo que hay solapamiento. Su llave primaria es el dni, que recibe como llave foránea de persona.
 - **Doctor** Esta base de datos está creada para Mediccenter SAC, que solo tiene un doctor por lo que el doctor no puede ser paciente u otro entidad, sin embargo hemos creado solapamiento para hacer la base de datos más escalable y que en un futuro sea más fácil agregar más doctores. La llave primaria del doctor es el dni de la persona, que recibe como llave foránea y tiene de atributo su sueldo.
 - **Interno** El interno puede ser en algún momento el paciente por lo que hay solapamiento. El interno recibe su llave primaria de persona, en forma de llave foránea. Además tiene como atributo su sueldo.
 - **Enfermera** Al igual que el doctor, solo hay una enfermera por lo que no puede haber solapamiento, pero se crea esta tabla en aras que la base de datos sea más escalable. Su llave primaria es el dni de persona, que está en calidad de llave foránea y tiene de atributo su sueldo.
 - **Gerente** En la tabla de Gerente también hay solapamiento, ya que el gerente puede ser el paciente. Su llave primaria es el dni de persona, que es la llave foránea. También tiene un atributo sueldo.
- **Superclase (Actividad Económica:)** La llave de Actividad Económica es compuesta y consta de su código y horaInicio, ello para que no haya dos actividades económicas a la misma hora. Tiene de atributo una descripción para la historia clínica y la fecha en la que se realiza dicha actividad, así como la duración de la actividad económica. Asimismo recibe como llave foránea el dni de Paciente, el dni de Doctor y el dni de Interno. Para esta tabla también se ha considerado solapamiento para hacer la base de datos más escalable.
- **Subclases**

- **Operación** La llave de operación es compuesta y consta del código de Actividad Económica y su horaInicio, que son foráneas, tiene de atributo el tipo de operación y el dni de la enfermera que está como llave foránea.
- **Consulta** Consulta es subclase de Actividad Económica, ya que tiene una llave compuesta por el código de Actividad Económica y de HoraInicio, de esta forma se garantiza que no haya dos consultas al mismo tiempo.

3.2.4. Relaciones binarias

- **detallePedido:** Esta relación se lleva a cabo entre Insumo médico y Pedido. Recibe como llave primaria y foránea la llave de Insumo medico y del pedido. Asimismo tiene el atributo de cantidad. Pedido tiene una multiplicidad de 1 a n puesto, ya que un pedido tiene que tener al menos un insumo medico. Insumo medico tiene una multiplicidad de 0 a n, ya que puede no aparecer en un pedido como si puede y en mayor cantidad.
- **utiliza** Esta relación se lleva a cabo entre Insumo médico y Operación. Tiene una llave compuesta por el código de operación y el número serial de Insumo médico, ambas llaves foráneas. Asimismo tiene un atributo de cantidad. Operación tiene una multiplicidad de 1 a n, ya que al menos utiliza un insumo médico. Insumo médico tiene una multiplicidad de de 0 a n, ya que puede no ser utilizado en una operación, pero también puede ser utilizado múltiples veces en una operación.

3.2.5. Relaciones ternarias

Reserva es una relación ternaria que involucra a Secretaria, Consulta y Paciente. Recibe como llave primaria y foránea la llave de consulta, ya que tiene una multiplicidad de 1 a 1. Paciente tiene una multiplicidad de 0 a n, ya que puede no reservar ni una cita, pero también puede reservar más de una. Secretaria tiene una multiplicidad de 1 a más, ya que tiene que al menos reservar una cita pero puede reservar más de una.

3.3. Diccionario de datos

3.3.1. Tabla de Persona

Nombre Campo	Tipo de dato	PK	FK	Descripción
DNI	VARCHAR(8)	X		DNI de la persona
NOMBRE	VARCHAR(25)			Nombre de la persona
APELLIDO	VARCHAR(25)			Apellido de la persona
FECHA DE NACIMIENTO	DATE			Fecha de nacimiento de la persona

Cuadro 4: Tabla de Persona

3.3.2. Tabla de Gerente

Nombre Campo	Tipo de dato	PK	FK	Descripción
P.DNI	VARCHAR(8)	X	X	DNI de la persona heredado hacia Gerente
SUELDO	DOUBLE			Sueldo del Gerente

Cuadro 5: Tabla de Gerente

3.3.3. Tabla de Enfermera

Nombre Campo	Tipo de dato	PK	FK	Descripción
P.DNI	VARCHAR(25)	X	X	DNI de la persona heredado hacia Enfermera
SUELDO	DOUBLE			Sueldo de la enfermera

Cuadro 6: Tabla de Enfermera

3.3.4. Tabla de Interno

Nombre Campo	Tipo de dato	PK	FK	Descripción
P.DNI	VARCHAR(8)	X	X	DNI de la persona heredado hacia Interno
SUELDO	DOUBLE			Sueldo del Interno

Cuadro 7: Tabla de Interno

3.3.5. Tabla de Doctor

Nombre Campo	Tipo de dato	PK	FK	Descripción
P.DNI	VARCHAR(8)	X	X	DNI de la persona heredado hacia Doctor
ESPECIALIDAD	VARCHAR(25)			Especialidad del Doctor
SUELDO	DOUBLE			Sueldo del Doctor

Cuadro 8: Tabla de Doctor

3.3.6. Tabla de Paciente

Nombre Campo	Tipo de dato	PK	FK	Descripción
P.DNI	VARCHAR(8)	X	X	DNI de la persona heredado hacia Paciente

Cuadro 9: Tabla de Paciente

3.3.7. Tabla de Secretaria

Nombre Campo	Tipo de dato	PK	FK	Descripción
P.DNI	VARCHAR(8)	X	X	DNI de la persona heredado hacia Secretaria
SUELDO	DOUBLE			Sueldo de la Secretaria

Cuadro 10: Tabla de Secretaria

3.3.8. Tabla de Pedido

Nombre Campo	Tipo de dato	PK	FK	Descripción
CÓDIGO	VARCHAR(10)	X		Código del pedido
GERENTE.DNI	VARCHAR(8)		X	DNI del Gerente como foreign key
FECHA	DATE			Fecha del pedido

Cuadro 11: Tabla de Pedido

3.3.9. Tabla de Insumo médico

Nombre Campo	Tipo de dato	PK	FK	Descripción
NÚMERO SERIAL	VARCHAR(10)	X		Número serial del Insumo médico
MATERIAL	VARCHAR(50)			Material del Insumo médico
MARCA	VARCHAR(30)			Marca del Insumo médico
COSTO POR UNIDAD	DOUBLE			Costo por unidad del Insumo médico
NOMBRE	VARCHAR(50)			Nombre del Insumo médico
CANTIDAD INVENTARIO	BIG INTEGER			Cantidad en el inventario del Insumo médico

Cuadro 12: Tabla de Insumo médico

3.3.10. Tabla de detalle Pedido

Nombre Campo	Tipo de dato	PK	FK	Descripción
INSUMO MÉDICO.NÚMERO SERIAL	VARCHAR(10)	X	X	Número serial del Insumo médico
PEDIDO.CÓDIGO	VARCHAR(10)			Código del pedido relacionado
CANTIDAD	BIG INTEGER			Cantidad del Insumo médico en el detalle Pedido

Cuadro 13: Tabla de detalle Pedido

3.3.11. Tabla de Actividad Económica

Nombre Campo	Tipo de dato	PK	FK	Descripción
PACIENTE.DNI	VARCHAR(8)		X	DNI del paciente relacionado
DOCTOR.DNI	VARCHAR(8)		X	DNI del doctor relacionado
INTERNO.DNI	VARCHAR(8)		X	DNI del interno relacionado
CÓDIGO	VARCHAR(10)	X		Código de la Actividad Económica
DESCRIPCIÓN	VARCHAR(250)			Descripción de la Actividad Económica

Cuadro 14: Tabla de Actividad Económica (continuación)

Nombre Campo	Tipo de dato	PK	FK	Descripción
FECHA	DATE			Fecha de la Actividad Económica

Cuadro 14: Tabla de Actividad Económica

3.3.12. Tabla de Consulta

Nombre Campo	Tipo de dato	PK	FK	Descripción
ACTIVIDAD ECONÓMICA.CÓDIGO	-	X	X	Código de la Actividad Económica
HORA INICIO	TIME	X		Hora de inicio de la Consulta
DURACIÓN	INTEGER			Duración de la Consulta

Cuadro 15: Tabla de Consulta

3.3.13. Tabla de Operación

Nombre Campo	Tipo de dato	PK	FK	Descripción
ACTIVIDAD ECONÓMICA.CÓDIGO	-	X	X	Código de la Actividad Económica
ENFERMERA.DNI	VARCHAR(8)		X	DNI de la Enfermera relacionada
TIPO DE OPERACIÓN	VARCHAR(50)			Fecha de la operación

Cuadro 16: Tabla de Operación

3.3.14. Tabla de Reserva

Nombre Campo	Tipo de dato	PK	FK	Descripción
CONSULTA.CÓDIGO	VARCHAR(10)	X	X	Código de la Consulta
PACIENTE.DNI	VARCHAR(8)		X	DNI del Paciente
SECRETARIA.DNI	VARCHAR(8)		X	DNI de la Secretaria

Cuadro 17: Tabla de Reserva (continuación)

Nombre Campo	Tipo de dato	PK	FK	Descripción
--------------	--------------	----	----	-------------

Cuadro 17: Tabla de Reserva

3.3.15. Tabla de Utiliza

Nombre Campo	Tipo de dato	PK	FK	Descripción
OPERACIÓN.CÓDIGO	VARCHAR(10)	X	X	Código de la Operación
INSUMO MÉDICO.NÚMERO SERIAL	VARCHAR(10)	X	X	Número serial del Insumo médico
CANTIDAD	BIG INTEGER			Cantidad del Insumo médico utilizado

Cuadro 18: Tabla de Utiliza

3.3.16. Tabla de Pago

Nombre Campo	Tipo de dato	PK	FK	Descripción
PAGO.CÓDIGO	VARCHAR(10)	X		Código del pago
ACTIVIDAD ECONÓMICA.CÓDIGO	VARCHAR(10)		X	Código de la Actividad Económica relacionada
FECHA	DATE			Fecha del pago
COSTO	DOUBLE			Costo del pago

Cuadro 19: Tabla de Pago

4. Implementación de la base de datos

4.1. Creación de Tablas en PostgreSQL

```
1 -- Creación de tablas
2
3     CREATE TABLE IF NOT EXISTS persona (
4         dni VARCHAR(8),
5         nombre VARCHAR(25),
6         apellido VARCHAR(25),
7         fecha_de_nacimiento DATE
8     );
9
10    CREATE TABLE IF NOT EXISTS gerente (
11        pdni VARCHAR(8),
12        sueldo DOUBLE PRECISION
13    );
14
15    CREATE TABLE IF NOT EXISTS secretaria (
16        pdni VARCHAR(8),
17        sueldo DOUBLE PRECISION
18    );
19
20    CREATE TABLE IF NOT EXISTS enfermera (
21        pdni VARCHAR(8),
22        sueldo DOUBLE PRECISION
23    );
24
25    CREATE TABLE IF NOT EXISTS interno (
26        pdni VARCHAR(8),
27        sueldo DOUBLE PRECISION
28    );
29
30    CREATE TABLE IF NOT EXISTS doctor (
31        pdni VARCHAR(8),
32        sueldo DOUBLE PRECISION,
33        especialidad VARCHAR(40)
34    );
35
36    CREATE TABLE IF NOT EXISTS paciente (
37        pdni VARCHAR(12)
38    );
39
40    CREATE TABLE IF NOT EXISTS pedido (
41        codigo VARCHAR(10),
42        gdni VARCHAR(8),
43        fecha_pedido DATE
44    );
45
46    CREATE TABLE IF NOT EXISTS insumo_medico (
47        nro_serial VARCHAR(10),
```

```

48     material VARCHAR(50),
49     marca VARCHAR(40),
50     costo_por_unidad DOUBLE PRECISION,
51     nombre VARCHAR(50),
52     cantidad INTEGER
53 );
54
55 CREATE TABLE IF NOT EXISTS detallePedido (
56     im_ns VARCHAR(10),
57     pcodigo VARCHAR(10),
58     cantidad INTEGER
59 );
60
61 CREATE TABLE IF NOT EXISTS actividad_economica (
62     padni VARCHAR(8),
63     ddni VARCHAR(8),
64     idni VARCHAR(8),
65     codigo VARCHAR(10),
66     descripcion VARCHAR(500),
67     fecha_ae DATE,
68     horaInicio TIME,
69     monto DOUBLE PRECISION
70 );
71
72 CREATE TABLE IF NOT EXISTS consulta (
73     aecodigo VARCHAR(10),
74     duracion INTEGER,
75     snrodoc VARCHAR(12)
76 );
77
78 CREATE TABLE IF NOT EXISTS operacion (
79     aecodigo VARCHAR(10),
80     edni VARCHAR(8),
81     tipo_operacion VARCHAR(50)
82 );
83
84 CREATE TABLE IF NOT EXISTS utiliza (
85     ocodigo VARCHAR(10),
86     im_ns VARCHAR(10),
87     cantidad INTEGER,
88     edni VARCHAR(8)
89 );
90
91 CREATE TABLE IF NOT EXISTS pago (
92     codigo VARCHAR(10),
93     aecodigo VARCHAR(10),
94     monto DOUBLE PRECISION,
95     tipo_de_pago VARCHAR(25),
96     fecha_pago DATE
97 );
98

```

```

99      -- Key constraints
100
101      -- Persona
102      ALTER TABLE persona ADD CONSTRAINT pk_persona_dni PRIMARY KEY (dni);
103
104      -- Gerente
105      ALTER TABLE gerente ADD CONSTRAINT pk_gerente_dni PRIMARY KEY (pdni);
106      ALTER TABLE gerente ADD CONSTRAINT fk_gerente_persona_dni
107          FOREIGN KEY (pdni) REFERENCES persona(dni);
108
109      -- Paciente
110      ALTER TABLE paciente ADD CONSTRAINT pk_paciente_dni PRIMARY KEY (pdni);
111      ALTER TABLE paciente ADD CONSTRAINT fk_paciente_persona_nrodoc
112          FOREIGN KEY (pdni) REFERENCES persona(dni);
113
114      -- Enfermera
115      ALTER TABLE enfermera ADD CONSTRAINT pk_enfermera_pdni PRIMARY KEY (
116          pdni);
117      ALTER TABLE enfermera ADD CONSTRAINT fk_enfermera_persona_nrodoc
118          FOREIGN KEY (pdni) REFERENCES persona(dni);
119
120      -- Interno
121      ALTER TABLE interno ADD CONSTRAINT pk_interno_pdni PRIMARY KEY (pdni);
122      ALTER TABLE interno ADD CONSTRAINT fk_interno_persona_nrodoc
123          FOREIGN KEY (pdni) REFERENCES persona(dni);
124
125      -- Doctor
126      ALTER TABLE doctor ADD CONSTRAINT pk_doctor_pdni PRIMARY KEY (pdni);
127      ALTER TABLE doctor ADD CONSTRAINT fk_doctor_persona_nrodoc
128          FOREIGN KEY (pdni) REFERENCES persona(dni);
129
130      -- Secretaria
131      ALTER TABLE secretaria ADD CONSTRAINT pk_secretaria_nrodoc PRIMARY KEY
132          (pdni);
133      ALTER TABLE secretaria ADD CONSTRAINT fk_secretaria_persona_pdni
134          FOREIGN KEY (pdni) REFERENCES persona(dni);
135
136      -- Pedido
137      ALTER TABLE pedido ADD CONSTRAINT pk_pedido_codigo PRIMARY KEY (codigo)
138      ;
139      ALTER TABLE pedido ADD CONSTRAINT fk_pedido_gerente_gdni
140          FOREIGN KEY (gdni) REFERENCES gerente(pdni);
141
142      -- Insumo m dico
143      ALTER TABLE insumo_medico ADD CONSTRAINT pk_insumomedico_ns PRIMARY KEY
144          (nro_serial);
145
146      -- detalle Pedido
147      ALTER TABLE detallePedido ADD CONSTRAINT pk_detallePedido PRIMARY KEY (
148          im_ns, pcodigo);

```

```

144    ALTER TABLE detallePedido ADD CONSTRAINT
145        fk_detallePedido_insumomedico_nroserial
146            FOREIGN KEY (im_ns) REFERENCES insumo_medico(nro_serial);
147    ALTER TABLE detallePedido ADD CONSTRAINT fk_detallePedido_pedido_codigo
148        FOREIGN KEY (pcodigo) REFERENCES pedido(codigo);

149    -- actividad_economica
150    ALTER TABLE actividad_economica ADD CONSTRAINT
151        pk_actividadEconomica_codigo PRIMARY KEY (codigo);
152    ALTER TABLE actividad_economica ADD CONSTRAINT
153        fk_actividadEconomica_paciente_nrodocumento
154            FOREIGN KEY (padni) REFERENCES paciente(pdni);
155    ALTER TABLE actividad_economica ADD CONSTRAINT
156        fk_actividadEconomica_doctor_nrodocumento
157            FOREIGN KEY (ddni) REFERENCES doctor(pdni);
158    ALTER TABLE actividad_economica ADD CONSTRAINT
159        fk_actividadEconomica_interno_nrodocumento
160            FOREIGN KEY (idni) REFERENCES interno(pdni);

161    -- consulta
162    ALTER TABLE consulta ADD CONSTRAINT fk_consulta_actividadEconomica
163        FOREIGN KEY (aecodigo)
164            REFERENCES actividad_economica(codigo);
165    ALTER TABLE consulta ADD CONSTRAINT pk_consulta_aecodigo
166        PRIMARY KEY (aecodigo);

167    -- operacion
168    ALTER TABLE operacion ADD CONSTRAINT fk_operacion_actividadEconomica
169        FOREIGN KEY (aecodigo) REFERENCES actividad_economica(codigo);
170    ALTER TABLE operacion ADD CONSTRAINT fk_operacion_enfermera_edni
171        FOREIGN KEY (edni) REFERENCES enfermera(pdni);
172    ALTER TABLE operacion ADD CONSTRAINT
173        pk_operacion_actividadEconomica_codigo
174            PRIMARY KEY (aecodigo);

175    -- utiliza
176    ALTER TABLE utiliza ADD CONSTRAINT fk_utiliza_operacion
177        FOREIGN KEY (ocodigo) REFERENCES operacion(aecodigo);
178    ALTER TABLE utiliza ADD CONSTRAINT fk_utiliza_enfermera_edni
179        FOREIGN KEY (edni) REFERENCES enfermera(pdni);
180    ALTER TABLE utiliza ADD CONSTRAINT fk_utiliza_insumoMedico_nroserial
181        FOREIGN KEY (im_ns) REFERENCES insumo_medico(nro_serial);
182    ALTER TABLE utiliza ADD CONSTRAINT pk_utiliza_ocodigo_edni_imns
183        PRIMARY KEY (ocodigo, edni, im_ns);

184    -- pago
185    ALTER TABLE pago ADD CONSTRAINT fk_pago_actividadEconomica
186        FOREIGN KEY (aecodigo) REFERENCES actividad_economica(codigo);
187    ALTER TABLE pago ADD CONSTRAINT pk_pago_codigo
188        PRIMARY KEY (codigo);

```

```

189 -- Unique Constraints
190
191 ALTER TABLE actividad_economica ADD CONSTRAINT unique_fechaae_horainicio
192 UNIQUE (fecha_ae, horainicio);
193
194 -- Not Null Constraints
195
196 ALTER TABLE persona ALTER COLUMN dni SET NOT NULL;
197 ALTER TABLE persona ALTER COLUMN nombre SET NOT NULL;
198 ALTER TABLE persona ALTER COLUMN apellido SET NOT NULL;
199
200 ALTER TABLE gerente ALTER COLUMN pdni SET NOT NULL;
201 ALTER TABLE gerente ALTER COLUMN sueldo SET NOT NULL;
202
203 ALTER TABLE secretaria ALTER COLUMN pdni SET NOT NULL;
204 ALTER TABLE secretaria ALTER COLUMN sueldo SET NOT NULL;
205
206 ALTER TABLE enfermera ALTER COLUMN pdni SET NOT NULL;
207 ALTER TABLE enfermera ALTER COLUMN sueldo SET NOT NULL;
208
209 ALTER TABLE interno ALTER COLUMN pdni SET NOT NULL;
210 ALTER TABLE interno ALTER COLUMN sueldo SET NOT NULL;
211
212 ALTER TABLE doctor ALTER COLUMN pdni SET NOT NULL;
213 ALTER TABLE doctor ALTER COLUMN sueldo SET NOT NULL;
214
215 ALTER TABLE paciente ALTER COLUMN pdni SET NOT NULL;
216
217 ALTER TABLE pedido ALTER COLUMN codigo SET NOT NULL;
218 ALTER TABLE pedido ALTER COLUMN gdni SET NOT NULL;
219
220 ALTER TABLE insumo_medico ALTER COLUMN nro_serial SET NOT NULL;
221 ALTER TABLE insumo_medico ALTER COLUMN costo_por_unidad SET NOT NULL;
222 ALTER TABLE insumo_medico ALTER COLUMN nombre SET NOT NULL;
223 ALTER TABLE insumo_medico ALTER COLUMN cantidad SET NOT NULL;
224
225 ALTER TABLE detallePedido ALTER COLUMN im_ns SET NOT NULL;
226 ALTER TABLE detallePedido ALTER COLUMN pcodigo SET NOT NULL;
227 ALTER TABLE detallePedido ALTER COLUMN cantidad SET NOT NULL;
228
229 ALTER TABLE actividad_economica ALTER COLUMN padni SET NOT NULL;
230 ALTER TABLE actividad_economica ALTER COLUMN ddni SET NOT NULL;
231 ALTER TABLE actividad_economica ALTER COLUMN idni SET NOT NULL;
232 ALTER TABLE actividad_economica ALTER COLUMN codigo SET NOT NULL;
233 ALTER TABLE actividad_economica ALTER COLUMN monto SET NOT NULL;
234
235 ALTER TABLE consulta ALTER COLUMN aecodigo SET NOT NULL;
236
237 ALTER TABLE operacion ALTER COLUMN aecodigo SET NOT NULL;
238 ALTER TABLE operacion ALTER COLUMN edni SET NOT NULL;
239

```

```

240 ALTER TABLE utiliza ALTER COLUMN ocodigo SET NOT NULL;
241 ALTER TABLE utiliza ALTER COLUMN im_ns SET NOT NULL;
242 ALTER TABLE utiliza ALTER COLUMN edni SET NOT NULL;
243
244 ALTER TABLE pago ALTER COLUMN codigo SET NOT NULL;
245 ALTER TABLE pago ALTER COLUMN aecodigo SET NOT NULL;
246
247 -- Stored Procedures & Triggers
248
249 -- Funci n y trigger para aumentar el stock de insumos m dicos cuando se
250 -- realiza un pedido
251 CREATE OR REPLACE FUNCTION aumentarStock() RETURNS TRIGGER AS $$ BEGIN
252     UPDATE insumo_medico
253     SET cantidad = cantidad + NEW.cantidad
254     WHERE nro_serial = NEW.im_ns;
255     RETURN NEW;
256 END;
257 $$ LANGUAGE plpgsql;
258
259 CREATE TRIGGER trg_aumentar_stock
260 AFTER INSERT ON detallePedido
261 FOR EACH ROW
262 EXECUTE FUNCTION aumentarStock();
263
264 -- Funci n y trigger para reducir el stock de insumos m dicos cuando se
265 -- utilizan
266 CREATE OR REPLACE FUNCTION reducirStock() RETURNS TRIGGER AS $$ BEGIN
267     UPDATE insumo_medico
268     SET cantidad = cantidad - NEW.cantidad
269     WHERE nro_serial = NEW.im_ns;
270     RETURN NEW;
271 END;
272 $$ LANGUAGE plpgsql;
273
274 CREATE TRIGGER trg_reducir_stock
275 AFTER INSERT ON utiliza
276 FOR EACH ROW
277 EXECUTE FUNCTION reducirStock();
278
279 CREATE OR REPLACE FUNCTION revisarMonto() RETURNS TRIGGER AS $$ DECLARE
280     monto_temporal DOUBLE PRECISION;
281 BEGIN
282     SELECT monto INTO monto_temporal
283     FROM actividad_economica
284     WHERE codigo = NEW.aecodigo;
285
286     IF FOUND THEN
287         IF NEW.monto != monto_temporal THEN

```

```

289      RAISE NOTICE 'El monto ingresado es diferente al monto de la
290      actividad economica';
291  END IF;
292 END IF;
293 RETURN NEW;
294END;
295$$ LANGUAGE plpgsql;
296
297CREATE TRIGGER revisar_monto
298AFTER INSERT ON pago
299FOR EACH ROW EXECUTE FUNCTION revisarMonto();

```

4.2. Carga de datos

Si bien seria más útil datos reales, lamentablemente no se pudo tener acceso. Para poder poner a prueba la base de datos y realizar querys, se pobló la base de datos con "dummy data."º data generada de forma aleatoria, respetando siempre la integridad de la base de datos, llaves primarias y llaves foráneas.

Para poblar la mayoría de la base de datos, se utilizó un script en python que utilizaba en su mayoria la libreria Faker. Este script se podrá encontrar en el anexo. Sin embargo, se tuvo problemas a la hora de generar la descripción de la tabla actividad-economica, ya que era un varchar(500). Por ello se generó vacío y se pobló dentro de PostgreSQL con el siguiente script:

```

1 CREATE OR REPLACE FUNCTION populate_descripcion()
2 RETURNS void AS $$$
3 DECLARE
4     rec RECORD;
5     random_text VARCHAR(500);
6     text_length INT := 100;
7 BEGIN
8     FOR rec IN SELECT * FROM actividad_economica LOOP
9         random_text := array_to_string(ARRAY(
10             SELECT
11                 CASE
12                     WHEN random() < 0.15 THEN ''
13                     ELSE chr((65 + round(random() * 25))::integer)
14                 END
15             FROM generate_series(1, text_length)
16         ), '');
17
18         UPDATE actividad_economica
19             SET descripcion = random_text
20             WHERE padni = rec.padni
21                 AND ddni = rec.ddni
22                 AND idni = rec.idni
23                 AND codigo = rec.codigo;
24     END LOOP;
25 END;

```

```
26 $$ LANGUAGE plpgsql;
```

Este script tiene que ser corrido en los 4 esquemas de la base de datos.

Para generar los datos en el script de python para los 4 esquemas, se cambia la cantidad de ceros en los fors de cada tabla. Finalmente, para subir los datos a PostgreSQL se utilizó el siguiente comando:

```
1 COPY <nombre_esquema>.<nombre_tabla> FROM '<direccion_csv>' DELIMITER ',', ';
```

en donde un ejemplo de uso sería así:

```
1 COPY millon.persona FROM 'D:\data_millon\personas_millon.csv' DELIMITER ',', ';
```

5. Optimización y Experimentación

El objetivo de esta sección es evaluar el rendimiento de la base de datos mediante tres consultas de complejidad media a alta. Las consultas se ejecutarán en diferentes esquemas con distintas cantidades de datos, tanto con índices como sin ellos, para comparar el rendimiento de la base de datos en ambas condiciones.

5.1. Consultas SQL para el Experimento

5.1.1. Descripción del Tipo de Consultas Seleccionadas

Se han seleccionado tres consultas de complejidad media a alta para evaluar el rendimiento de la base de datos en términos de tiempo de ejecución y manejo de datos. Las consultas seleccionadas son:

- **Porcentaje de costo de insumos médicos:** Calcula el porcentaje que representa la suma de los costos de todos los insumos médicos utilizados en una operación y de ahí calcula el promedio de todos los porcentajes de las operaciones.

Justificación: Se quiere saber cuánto representa el costo de los insumos médicos de las operaciones, para determinar si se le debe de subir el precio para así aumentar las ganancias o en caso los insumos médicos representen una porción considerable de la operación, disminuir su costo.

- **Reporte sobre tipo de operaciones y cantidad de personas involucradas:** Agrupa todos los tipos de operaciones de hay y muestra cuántas operaciones de ese tipo se han hecho, el costo total de los insumos médicos y la cantidad de enfermeras y doctores distintos que han participado en ese tipo de operaciones.

Justificación: Se quiere tener un informe detallado sobre la cantidad de operaciones que se hacen por cada tipo y cuántas personas están involucradas.

- **Informe de operaciones de doctores:** Busca todas las operaciones hechas por doctores que fueron pagadas con efectivo y las agrupa por el dni del doctor. Además muestra el monto del pago y la fecha.

Justificación: Se quiere saber que doctores son los que más reciben pagos en efectivo para quizas crear correlaciones.

5.1.2. Implementación de Consultas en SQL

Consulta 1: Porcentaje de costo de insumos médicos en insumos medicos on un costo mayor a 95 por unidad

```

1 EXPLAIN ANALYZE
2
3 SELECT
4   AVG(porcentaje_costo_insumos) AS promedio_porcentajes
5 FROM (
6   SELECT
7     (SUM(u.cantidad * i.costo_por_unidad) / a.monto) * 100 AS
8       porcentaje_costo_insumos
9   FROM utiliza u
10  JOIN insumo_medico i
11    ON u.im_ns = i.nro_serial
12  JOIN operacion o
13    ON o.aecodigo = u.ocodigo
14  JOIN actividad_economica a
15    ON a.codigo = o.aecodigo
16 WHERE
17   i.costo_por_unidad > 95
18 GROUP BY o.aecodigo, a.monto
) subquery;

```

Consulta 2: Reporte sobre tipo de operaciones y cantidad de personas involucradas

```

1 EXPLAIN ANALYZE
2
3 SELECT
4   o.tipo_operacion,
5   COUNT(o.aecodigo) AS total_operaciones,
6   SUM(im.costo_por_unidad * u.cantidad) AS costo_total_insumos,
7   COUNT(DISTINCT e.pdni) AS total_enfermeras,
8   COUNT(DISTINCT d.pdni) AS total_doctores
9
10  FROM operacion o
11  JOIN actividad_economica ae ON o.aecodigo = ae.codigo
12  JOIN utiliza u ON o.aecodigo = u.ocodigo
13  JOIN insumo_medico im ON u.im_ns = im.nro_serial
14  JOIN enfermera e ON u.edni = e.pdni
15  JOIN doctor d ON ae.ddni = d.pdni
16 WHERE e.sueldo > 5500
17 AND d.sueldo > 250000
18 GROUP BY o.tipo_operacion
19 ORDER BY total_operaciones DESC;

```

Consulta 3: Doctores que han recibido pagos en efectivo, la descripción de la actividad económica y el monto de la misma

```
1 EXPLAIN ANALYZE
2
3 SELECT
4     p.nombre,
5     p.apellido,
6     d.especialidad,
7     ae.descripcion,
8     pg.monto,
9     pg.fecha_pago
10
11    FROM
12        persona p
13    JOIN
14        doctor d ON p.dni = d.pdni
15    JOIN
16        actividad_economica ae ON d.pdni = ae.ddni
17    JOIN
18        pago pg ON ae.codigo = pg.aecodigo
19 WHERE
20     pg.tipo_de_pago = 'Efectivo'
21 GROUP BY d.pdni, p.nombre, p.apellido, d.especialidad, ae.descripcion, pg.
22         monto, pg.fecha_pago
```

5.2. Metodología del experimento

5.2.1. Experimentación sin índices

Para la experimentación sin índices se ejecutó las siguientes líneas para que el PostgreSQL no agregue índices indeseados.

```
1 SET enable_mergejoin TO OFF;
2 SET enable_hashjoin TO OFF;
3 SET enable_bitmapscan TO OFF ;
4 SET enable_sort TO OFF;
```

Asimismo, antes de ejecutar cada query, se ejecutaba el comando **VACUUM FULL tabla1, tabla2, ...** con las tablas participes de la query.

Por último, se utilizó **EXPLAIN ANALYZE** para poder saber query plan de las queries y la duración en milisegundos de la misma.

5.2.2. Experimentación con índices

Para la experimentación con índices se ejecutó el siguiente comando de líneas para que se utilicen los índices

```
1 SET enable_mergejoin TO ONN;
2 SET enable_hashjoin TO ONN;
3 SET enable_bitmapscan TO ONN ;
4 SET enable_sort TO ONN;
```

Asimismo, antes de ejecutar cada query, se ejecutaba el comando **VACUUM FULL tabla1, tabla2, ...** con las tablas partícipes de la query.

Por último, se utilizó **EXPLAIN ANALYZE** para poder saber query plan de las querys y la duracion en milisegundos de la misma.

Adicionalmente, durante la experimentación de querys con índices, se agrego y quitó índices a varios atributos, así como eliminar primary keys para poder forzar índices. Esto con el objetivo de probar diferentes escenarios, para poder saber cual es la forma más eficiente de optimizar nuestras querys.

5.3. Optimización de consultas

5.3.1. Planes de índices para Consulta 1

En un principio se decidió probar con índices en el atributo **costo-por-unidad** de insumo-medico. Al probar con un índice hash, el pgadmin no lo utilizaba, sin embargo al colocar un índice btree, el pgadmin dejó de hacer una Parallel Seq Scan y pasó a hacer un bitmap index scan, utilizando el índice agregado.

```
1 CREATE INDEX idx_insumo_medico_costo_por_unidad ON insumo_medico USING
  HASH (costo_por_unidad);
2 CREATE INDEX idx_insumo_medico_costo_por_unidad ON insumo_medico USING
  BTREE (costo_por_unidad);
```

También se pensó en agregar un índice en im-ns, de la relación utiliza. Para ello se tuvo que descartar una primary key y dos foreign keys.

```
1 ALTER TABLE insumo_medico DROP CONSTRAINT pk_insumomedico_ns
2 ALTER TABLE detallePedido DROP CONSTRAINT
  fk_detallepedido_insumomedico_nroserial
3 ALTER TABLE utiliza DROP CONSTRAINT fk_utiliza_insumomedico_nroserial
```

De ahí se agrego un índice en utiliza.im-ns. Además se propuso agregar un índice en utiliza.ocodigo por lo que se tuvo que eliminar más constraints

```
1 ALTER TABLE operacion ADD CONSTRAINT
  pk_operacion_actividadEconomica_codigo
  PRIMARY KEY (aecodigo);
2 ALTER TABLE utiliza ADD CONSTRAINT fk_utiliza_operacion
  FOREIGN KEY (ocodigo) REFERENCES operacion(aecodigo);
```

```
1 CREATE INDEX idx_utiliza_nro_serial ON utiliza USING BTREE (im_ns);
2 CREATE INDEX idx_utiliza_ocodigo ON utiliza USING BTREE (ocodigo)
```

El resultado de esta query fue prácticamente idéntico al de la query con solo un índice en costo-por-unidad, por lo que se decidió mantener la query con solo un índice. Ello porque utilizar la query que elimina constraints implicaría desnaturalizar las tablas y base de datos y para mantener la integridad de los datos y de la base de datos, se tendrían que

agregar nuevamente esos constraints. Por lo tanto, si se considera el tiempo de eliminar y crear los constraints, esta proposición de query no es viable.

En conclusión, el índice propuesto para la primera consulta es:

```
1 CREATE INDEX idx_insumo_medico_costo_por_unidad ON insumo_medico USING  
BTREE (costo_por_unidad);
```

5.3.2. Planes de índices para Consulta 2

Para la segunda consulta se consideró agregar índices en los sueldos de enfermera y doctor. Se probó tanto con índices BTREE y HASH, sin embargo los índices BTREE eran los únicos identificados.

```
1 CREATE INDEX idx_sueldo_doctor ON doctor USING BTREE (sueldo);  
2 CREATE INDEX idx_sueldo_enfermera ON enfermera USING BTREE (sueldo);
```

El análisis y discusión del porqué, se hará en su respectivo inciso.

Al igual que en la primera consulta, se propuso la idea de eliminar constraints y agregar índices adicionales en utiliza.ocodigo y utiliza.in-ns. Sin embargo, utilizando la información de la consulta 1 de ayuda, se determinó no agregar dichos índices. Por lo tanto, la planificación de índices para la segunda consulta sería la siguiente:

```
1 CREATE INDEX idx_sueldo_doctor ON doctor USING BTREE (sueldo);  
2 CREATE INDEX idx_sueldo_enfermera ON enfermera USING BTREE (sueldo);
```

5.3.3. Planes de índices para Consulta 3

Para la tercera consulta se decidió usar un índice en el atributo tipo-de-pago de pago.

```
1 CREATE INDEX idx_tipo_de_pago ON pago USING btree (tipo_de_pago)
```

5.4. Plataforma de Pruebas

Sistema Operativo	Windows 11 64-bits
RAM	16 GB
CPU	AMD Ryzen 7 4800 H
Capacidad SSD	1 TB
PostgreSQL	16.2

5.5. Medición de tiempos

5.5.1. Ejecución de consulta 1

QUERY PLAN	
	text
1	Aggregate (cost=236.43..236.44 rows=1 width=8) (actual time=2.331..2.333 rows=1 loops=1)
2	-> HashAggregate (cost=230.10..233.55 rows=230 width=26) (actual time=2.286..2.317 rows=227 loops=1)
3	Group Key: o.aecodigo, a.monto
4	Batches: 1 Memory Usage: 64kB
5	-> Hash Join (cost=203.18..227.23 rows=230 width=30) (actual time=2.029..2.212 rows=257 loops=1)
6	Hash Cond: ((o.aecodigo)::text = (u.ocodigo)::text)
7	-> Seq Scan on operacion o (cost=0.00..18.00 rows=1000 width=10) (actual time=0.008..0.064 rows=1000 loops=1)
8	-> Hash (cost=200.30..200.30 rows=230 width=40) (actual time=1.979..1.981 rows=257 loops=1)
9	Buckets: 1024 Batches: 1 Memory Usage: 27kB
10	-> Nested Loop (cost=17.06..200.30 rows=230 width=40) (actual time=0.088..1.920 rows=257 loops=1)
11	-> Hash Join (cost=16.78..121.96 rows=230 width=22) (actual time=0.067..0.816 rows=257 loops=1)
12	Hash Cond: ((u.im_ns)::text = (i.nro_serial)::text)
13	-> Seq Scan on utiliza u (cost=0.00..92.00 rows=5000 width=24) (actual time=0.006..0.282 rows=5000 loops=1)
14	-> Hash (cost=16.21..16.21 rows=46 width=18) (actual time=0.054..0.054 rows=46 loops=1)
15	Buckets: 1024 Batches: 1 Memory Usage: 11kB
16	-> Bitmap Heap Scan on insumo_medico i (cost=4.63..16.21 rows=46 width=18) (actual time=0.029..0.045 rows=48 loops=1)
17	Recheck Cond: (costo_por_unidad > '95)::double precision
18	Heap Blocks: exact=11
19	-> Bitmap Index Scan on idx_insumo_medico_costo_por_unidad (cost=0.00..4.62 rows=46 width=0) (actual time=0.025..0.025 rows=48 loops=1)
20	Index Cond: (costo_por_unidad > '95)::double precision
21	-> Index Scan using pk_actividadeconomico_codigo on actividad_economica a (cost=0.28..0.34 rows=1 width=18) (actual time=0.004..0.004 rows=1 loops=1)
22	Index Cond: ((codigo)::text = (u.ocodigo)::text)
23	Planning Time: 7.509 ms
24	Execution Time: 2.401 ms

Figura 1: Query Plan para la consulta 1 con índices en la tabla de 1 mil

QUERY PLAN	
text	
1	Aggregate (cost=680.66..680.67 rows=1 width=8) (actual time=8.454..8.455 rows=1 loops=1)
2	-> HashAggregate (cost=674.33..677.78 rows=230 width=26) (actual time=8.402..8.438 rows=227 loops=1)
3	Group Key: o.aecodigo, a.monto
4	Batches: 1 Memory Usage: 64kB
5	-> Nested Loop (cost=0.84..671.46 rows=230 width=30) (actual time=0.265..8.282 rows=257 loops=1)
6	-> Nested Loop (cost=0.56..601.61 rows=230 width=40) (actual time=0.111..7.084 rows=257 loops=1)
7	-> Nested Loop (cost=0.29..523.27 rows=230 width=22) (actual time=0.087..5.848 rows=257 loops=1)
8	-> Seq Scan on utiliza u (cost=0.00..92.00 rows=5000 width=24) (actual time=0.012..0.323 rows=5000 loops=1)
9	-> Memoize (cost=0.29..0.32 rows=1 width=18) (actual time=0.001..0.001 rows=0 loops=5000)
10	Cache Key: u.im_ns
11	Cache Mode: logical
12	Hits: 4008 Misses: 992 Evictions: 0 Overflows: 0 Memory Usage: 75kB
13	-> Index Scan using pk_insumomedico_ns on insumo_medico i (cost=0.28..0.31 rows=1 width=18) (actual time=0.004..0.004 rows=0 loops=992)
14	Index Cond: ((nro_serial)::text = (u.im_ns)::text)
15	Filter: (costo_por_unidad > '95'::double precision)
16	Rows Removed by Filter: 1
17	-> Index Scan using pk_actividadeconomico_codigo on actividad_economica a (cost=0.28..0.34 rows=1 width=18) (actual time=0.004..0.004 rows=1 loops=1)
18	Index Cond: ((codigo)::text = (u.ocodigo)::text)
19	-> Index Only Scan using pk_operacion_actividadeconomico_codigo on operacion o (cost=0.28..0.30 rows=1 width=10) (actual time=0.004..0.004 rows=1 loops=1)
20	Index Cond: (aecodigo = (u.ocodigo)::text)
21	Heap Fetches: 257
22	Planning Time: 6.137 ms
23	Execution Time: 8.615 ms

Figura 2: Query Plan para la consulta 1 sin índices en la tabla de 1 mil

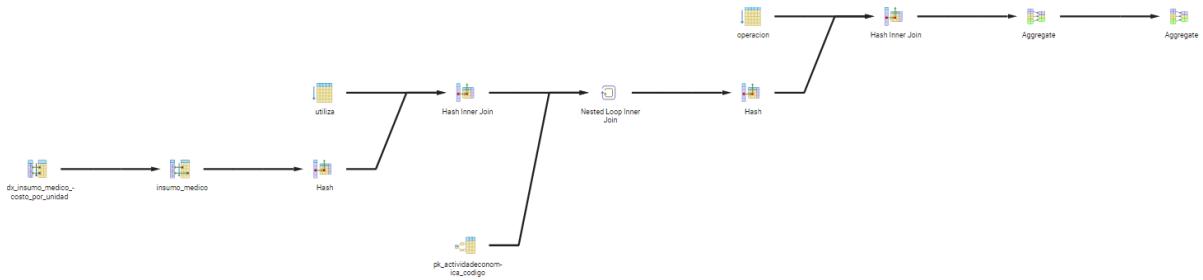


Figura 3: Query Plan Tree para la planificación con índices en la tabla de 1 mil

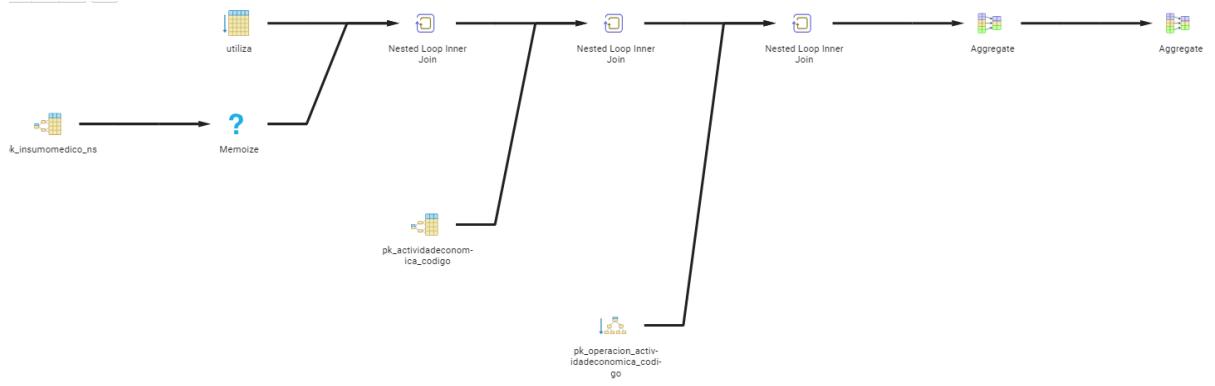


Figura 4: Query Plan Tree para la planificación sin índices en la tabla de 1 mil

	QUERY PLAN text
1	Aggregate (cost=2400.19..2400.20 rows=1 width=8) (actual time=16.738..16.743 rows=1 loops=1)
2	-> HashAggregate (cost=2325.53..2366.25 rows=2715 width=26) (actual time=16.174..16.611 rows=2375 loops=1)
3	Group Key: o.aecodigo, a.monto
4	Batches: 1 Memory Usage: 369kB
5	-> Hash Join (cost=1523.44..2291.59 rows=2715 width=30) (actual time=11.023..15.463 rows=2704 loops=1)
6	Hash Cond: ((a.codigo)::text = (o.aecodigo)::text)
7	-> Seq Scan on actividad_economica a (cost=0.00..666.00 rows=20000 width=18) (actual time=0.008..1.420 rows=20000 loops=1)
8	-> Hash (cost=1489.50..1489.50 rows=2715 width=32) (actual time=11.008..11.011 rows=2704 loops=1)
9	Buckets: 4096 Batches: 1 Memory Usage: 209kB
10	-> Hash Join (cost=434.07..1489.50 rows=2715 width=32) (actual time=2.205..10.549 rows=2704 loops=1)
11	Hash Cond: ((u.ocodigo)::text = (o.aecodigo)::text)
12	-> Hash Join (cost=136.07..1184.37 rows=2715 width=22) (actual time=0.419..7.903 rows=2704 loops=1)
13	Hash Cond: ((u.im_ns)::text = (i.nro_serial)::text)
14	-> Seq Scan on utiliza u (cost=0.00..917.00 rows=50000 width=24) (actual time=0.006..2.741 rows=50000 loops=1)
15	-> Hash (cost=129.28..129.28 rows=543 width=18) (actual time=0.360..0.361 rows=548 loops=1)
16	Buckets: 1024 Batches: 1 Memory Usage: 36kB
17	-> Bitmap Heap Scan on insumo_medico i (cost=12.49..129.28 rows=543 width=18) (actual time=0.072..0.289 rows=548 loops=1)
18	Recheck Cond: (costo_por_unidad > '95)::double precision
19	Heap Blocks: exact=109
20	-> Bitmap Index Scan on idx_insumo_medico_costo_por_unidad (cost=0.00..12.36 rows=543 width=0) (actual time=0.058..0.058 rows=548 loops=1)
21	Index Cond: (costo_por_unidad > '95)::double precision
22	-> Hash (cost=173.00..173.00 rows=10000 width=10) (actual time=1.768..1.768 rows=10000 loops=1)
23	Buckets: 16384 Batches: 1 Memory Usage: 547kB
24	-> Seq Scan on operacion o (cost=0.00..173.00 rows=10000 width=10) (actual time=0.017..0.732 rows=10000 loops=1)
25	Planning Time: 8.032 ms
26	Execution Time: 17.022 ms

Figura 5: Query Plan para la consulta 1 con índices en la tabla de 10 mil

QUERY PLAN text	
1	Aggregate (cost=7146.14..7146.15 rows=1 width=8) (actual time=146.888..146.891 rows=1 loops=1)
2	-> HashAggregate (cost=7071.48..7112.20 rows=2715 width=26) (actual time=145.964..146.679 rows=2375 loops=1)
3	Group Key: o.aecodigo, a.monto
4	Batches: 1 Memory Usage: 369kB
5	-> Nested Loop (cost=0.87..7037.54 rows=2715 width=30) (actual time=0.392..143.912 rows=2704 loops=1)
6	-> Nested Loop (cost=0.58..6191.49 rows=2715 width=40) (actual time=0.210..123.194 rows=2704 loops=1)
7	-> Nested Loop (cost=0.30..5245.04 rows=2715 width=22) (actual time=0.185..99.613 rows=2704 loops=1)
8	-> Seq Scan on utiliza u (cost=0.00..917.00 rows=50000 width=24) (actual time=0.025..4.989 rows=50000 loops=1)
9	-> Memoize (cost=0.30..0.33 rows=1 width=18) (actual time=0.002..0.002 rows=0 loops=50000)
10	Cache Key: u.im_ns
11	Cache Mode: logical
12	Hits: 40074 Misses: 9926 Evictions: 0 Overflows: 0 Memory Usage: 753kB
13	-> Index Scan using pk_insumomedico_ns on insumo_medico i (cost=0.29..0.32 rows=1 width=18) (actual time=0.006..0.006 rows=0 loops=9926)
14	Index Cond: ((nro_serial)::text = (u.im_ns)::text)
15	Filter: (costo_por_unidad > '95'::double precision)
16	Rows Removed by Filter: 1
17	-> Index Scan using pk_actividadeconomica_codigo on actividad_economica a (cost=0.29..0.35 rows=1 width=18) (actual time=0.008..0.008 rows=1 loops=1)
18	Index Cond: ((codigo)::text = (u.ocodigo)::text)
19	-> Index Only Scan using pk_operacion_actividadeconomica_codigo on operacion o (cost=0.29..0.31 rows=1 width=10) (actual time=0.007..0.007 rows=1 loops=1)
20	Index Cond: (aecodigo = (u.ocodigo)::text)
21	Heap Fetches: 2704
22	Planning Time: 6.087 ms
23	Execution Time: 147.543 ms

Figura 6: Query Plan para la consulta 1 sin índices en la tabla de 10 mil

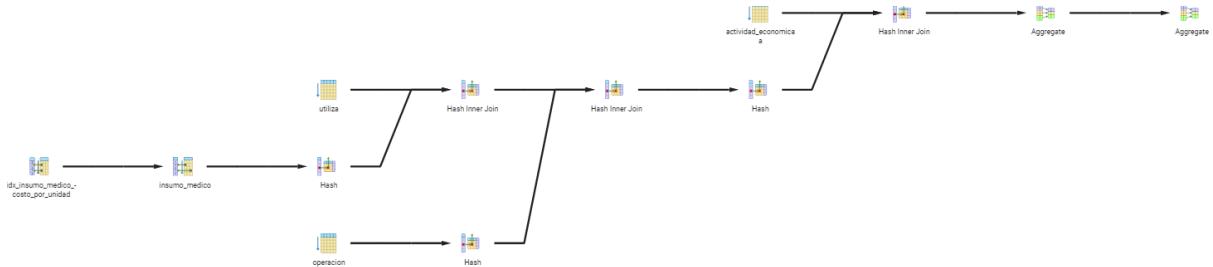


Figura 7: Query Plan Tree para la planificación con índices en la tabla de 10 mil

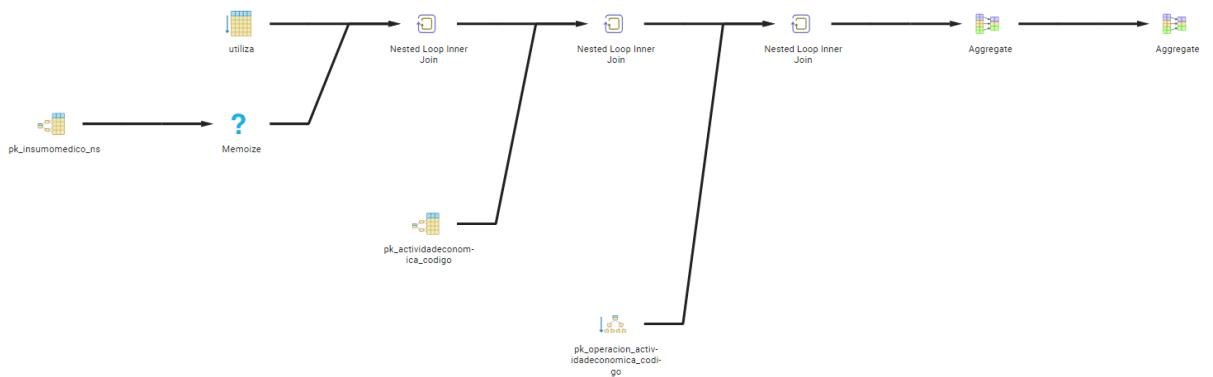


Figura 8: Query Plan Tree para la planificación sin índices en la tabla de 10 mil

QUERY PLAN	
text	
1	Aggregate (cost=21067.11..21067.12 rows=1 width=8) (actual time=289.084..289.477 rows=1 loops=1)
2	-> Finalize HashAggregate (cost=20311.27..20723.55 rows=27485 width=26) (actual time=280.725..287.662 rows=24462 loops=1)
3	Group Key: o.aecodigo, a.monto
4	Batches: 1 Memory Usage: 4881kB
5	-> Gather (cost=17734.57..20139.49 rows=22904 width=26) (actual time=263.599..270.538 rows=26803 loops=1)
6	Workers Planned: 2
7	Workers Launched: 2
8	-> Partial HashAggregate (cost=16734.57..16849.09 rows=11452 width=26) (actual time=231.941..234.072 rows=8934 loops=3)
9	Group Key: o.aecodigo, a.monto
10	Batches: 1 Memory Usage: 1425kB
11	Worker 0: Batches: 1 Memory Usage: 1425kB
12	Worker 1: Batches: 1 Memory Usage: 1425kB
13	-> Hash Join (cost=4257.28..16591.42 rows=11452 width=30) (actual time=34.612..224.897 rows=9369 loops=3)
14	Hash Cond: ((u.ocodigo)::text = (o.aecodigo)::text)
15	-> Nested Loop (cost=1286.28..13590.36 rows=11452 width=40) (actual time=2.770..185.597 rows=9369 loops=3)
16	-> Parallel Hash Join (cost=1285.86..8083.09 rows=11452 width=22) (actual time=1.646..68.689 rows=9369 loops=3)
17	Hash Cond: ((u.im_ns)::text = (i.nro_serial)::text)
18	-> Parallel Seq Scan on utiliza u (cost=0.00..6250.33 rows=208333 width=24) (actual time=0.473..28.676 rows=166667 loops=3)
19	-> Parallel Hash (cost=1245.44..1245.44 rows=3234 width=18) (actual time=1.095..1.097 rows=1865 loops=3)
20	Buckets: 8192 Batches: 1 Memory Usage: 384kB
21	-> Parallel Bitmap Heap Scan on insumo_medico i (cost=107.02..1245.44 rows=3234 width=18) (actual time=0.564..2.363 rows=5595 loops=1)
22	Recheck Cond: (costo_por_unidad > 95::double precision)
23	Heap Blocks: exact=1090
24	-> Bitmap Index Scan on idx_insumo_medico_costo_por_unidad (cost=0.00..105.64 rows=5497 width=0) (actual time=0.456..0.456 rows=5595 loops=1)
25	Index Cond: (costo_por_unidad > 95::double precision)
26	-> Index Scan using pk_actividad_economica_codigo on actividad_economica a (cost=0.42..0.48 rows=1 width=18) (actual time=0.012..0.012 rows=1 loops=1)
27	Index Cond: ((codigo)::text = (u.ocodigo)::text)
28	-> Hash (cost=1721.00..1721.00 rows=100000 width=10) (actual time=31.645..31.645 rows=100000 loops=3)
29	Buckets: 131072 Batches: 1 Memory Usage: 5213kB
30	-> Seq Scan on operacion o (cost=0.00..1721.00 rows=100000 width=10) (actual time=0.479..10.353 rows=100000 loops=3)
31	Planning Time: 9.472 ms
32	Execution Time: 290.694 ms
Total rows: 32 of 32 Query complete 00:00:00.336	

Figura 9: Query Plan para la consulta 1 con índices en la tabla de 100 mil (primera parte)

QUERY PLAN	
text	
1	Aggregate (cost=82956.37..82956.38 rows=1 width=8) (actual time=1165.906..1165.978 rows=1 loops=1)
2	-> Finalize HashAggregate (cost=82200.53..82612.81 rows=27485 width=26) (actual time=1158.114..1164.433 rows=24462 loops=1)
3	Group Key: o.aecodigo, a.monto
4	Batches: 1 Memory Usage: 3089kB
5	-> Gather (cost=79623.83..82028.75 rows=22904 width=26) (actual time=1143.089..1149.147 rows=26787 loops=1)
6	Workers Planned: 2
7	Workers Launched: 2
8	-> Partial HashAggregate (cost=78623.83..78738.35 rows=11452 width=26) (actual time=1115.037..1117.013 rows=8929 loops=3)
9	Group Key: o.aecodigo, a.monto
10	Batches: 1 Memory Usage: 1425kB
11	Worker 0: Batches: 1 Memory Usage: 1425kB
12	Worker 1: Batches: 1 Memory Usage: 1425kB
13	-> Nested Loop (cost=1.26..78480.68 rows=11452 width=30) (actual time=2.749..1106.454 rows=9369 loops=3)
14	-> Nested Loop (cost=0.85..73397.46 rows=11452 width=40) (actual time=1.863..1011.510 rows=9369 loops=3)
15	-> Nested Loop (cost=0.43..67890.19 rows=11452 width=22) (actual time=0.762..890.997 rows=9369 loops=3)
16	-> Parallel Seq Scan on <i>utiliza u</i> (cost=0.00..6250.33 rows=208333 width=24) (actual time=0.473..40.128 rows=166667 loops=3)
17	-> Memoize (cost=0.43..0.46 rows=1 width=18) (actual time=0.005..0.005 rows=0 loops=50000)
18	Cache Key: u.im_ns
19	Cache Mode: logical
20	Hits: 91570 Misses: 82510 Evictions: 0 Overflows: 0 Memory Usage: 6264kB
21	Worker 0: Hits: 85551 Misses: 81009 Evictions: 0 Overflows: 0 Memory Usage: 6152kB
22	Worker 1: Hits: 79547 Misses: 79813 Evictions: 0 Overflows: 0 Memory Usage: 6057kB
23	-> Index Scan using <i>pk_insumomedico_ns</i> on <i>insumo_medico i</i> (cost=0.42..0.45 rows=1 width=18) (actual time=0.009..0.009 rows=0 loops=243332)
24	Index Cond: ((nro_serial)::text = (u.im_ns)::text)
25	Filter: (costo_por_unidad > '95'::double precision)
26	Rows Removed by Filter: 1
27	-> Index Scan using <i>pk_actividadeconomic_a</i> on <i>actividad_economica a</i> (cost=0.42..0.48 rows=1 width=18) (actual time=0.012..0.012 rows=1 loops=2...)
28	Index Cond: ((codigo)::text = (u.ocodigo)::text)
29	-> Index Only Scan using <i>pk_operacion_actividadeconomico_codigo</i> on <i>operacion o</i> (cost=0.42..0.44 rows=1 width=10) (actual time=0.010..0.010 rows=1 loops=...)
30	Index Cond: (aecodigo = (u.ocodigo)::text)
30	Index Cond: (aecodigo = (u.ocodigo)::text)
31	Heap Fetches: 28108
32	Planning Time: 9.181 ms
33	Execution Time: 1167.438 ms
Total rows: 33 of 33 Query complete 00:00:01.211	

Figura 10: Query Plan para la consulta 1 sin índices en la tabla de 100 mil

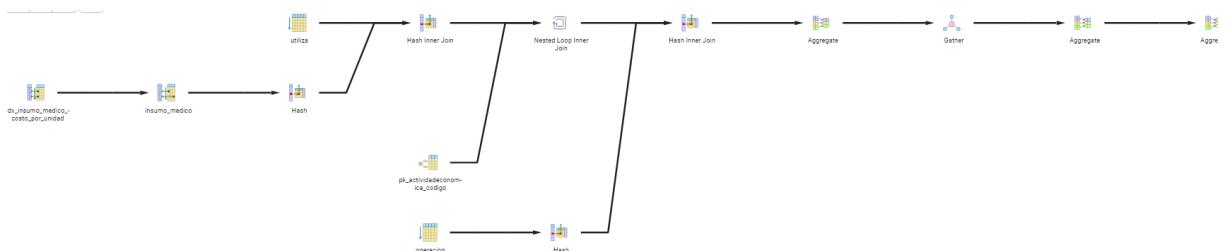


Figura 11: Query Plan Tree para la planificación con índices en la tabla de 100 mil

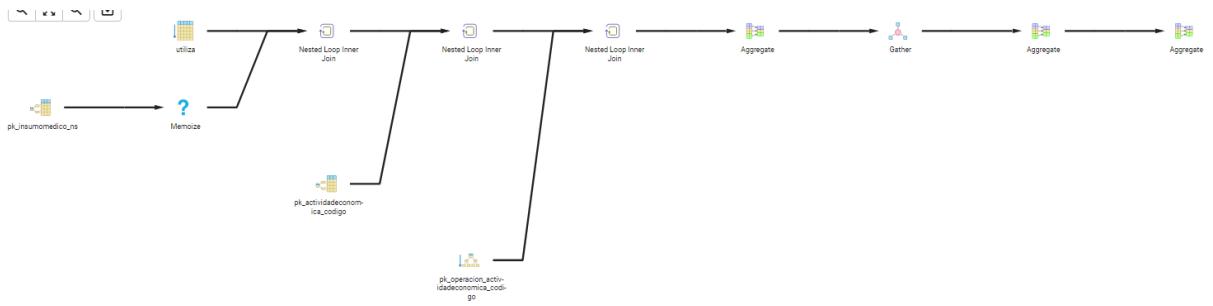


Figura 12: Query Plan Tree para la planificación sin índices en la tabla de 100 mil

QUERY PLAN text	
1	Aggregate (cost=190732.03..190732.04 rows=1 width=8) (actual time=3708.761..3715.073 rows=1 loops=1)
2	-> Finalize HashAggregate (cost=182880.09..187162.97 rows=285525 width=26) (actual time=3630.795..3698.570 rows=243082 loops=1)
3	Group Key: o.aecodigo, a.monto
4	Batches: 1 Memory Usage: 36881kB
5	-> Gather (cost=156112.07..181095.56 rows=237938 width=26) (actual time=3501.533..3552.393 rows=243082 loops=1)
6	Workers Planned: 2
7	Workers Launched: 2
8	-> Partial HashAggregate (cost=155112.07..156301.76 rows=118969 width=26) (actual time=3463.471..3484.010 rows=81027 loops=3)
9	Group Key: o.aecodigo, a.monto
10	Batches: 1 Memory Usage: 14353kB
11	Worker 0: Batches: 1 Memory Usage: 14353kB
12	Worker 1: Batches: 1 Memory Usage: 14353kB
13	-> Parallel Hash Join (cost=140197.09..153624.95 rows=118969 width=30) (actual time=3227.896..3422.239 rows=92900 loops=3)
14	Hash Cond: ((o.aecodigo)::text = (u.ocodigo)::text)
15	-> Parallel Seq Scan on operacion o (cost=0.00..11369.67 rows=416667 width=10) (actual time=0.824..50.813 rows=333333 loops=3)
16	-> Parallel Hash (cost=138709.97..138709.97 rows=118969 width=40) (actual time=3225.716..3225.722 rows=92900 loops=3)
17	Buckets: 524288 Batches: 1 Memory Usage: 25952kB
18	-> Nested Loop (cost=12640.26..138709.97 rows=118969 width=40) (actual time=57.668..3153.525 rows=92900 loops=3)
19	-> Parallel Hash Join (cost=12639.84..80608.94 rows=118969 width=22) (actual time=56.659..875.842 rows=92900 loops=3)
20	Hash Cond: ((u.im_ns)::text = (i.nro_serial)::text)
21	-> Parallel Seq Scan on insumo_medico u (cost=0.00..62500.33 rows=2083333 width=24) (actual time=0.725..243.836 rows=1666667 loops=3)
22	-> Parallel Hash (cost=12342.41..12342.41 rows=23794 width=18) (actual time=55.796..55.797 rows=18575 loops=3)
23	Buckets: 65536 Batches: 1 Memory Usage: 3616kB
24	-> Parallel Bitmap Heap Scan on insumo_medico i (cost=1070.99..12342.41 rows=23794 width=18) (actual time=1.970..50.116 rows=18575 loops=3)
25	Recheck Cond: (costo_por_unidad > '95)::double precision
26	Heap Blocks: exact=7584
27	-> Bitmap Index Scan on idx_insumo_medico_costo_por_unidad (cost=0.00..1056.71 rows=57105 width=0) (actual time=4.568..4.568 rows=55724 loops=3)
28	Index Cond: (costo_por_unidad > '95)::double precision
29	-> Index Scan using pk_actividadeconomica_codigo on actividad_economica a (cost=0.43..0.49 rows=1 width=18) (actual time=0.024..0.024 rows=1 loops=278...)
30	Index Cond: ((codigo)::text = (u.ocodigo)::text)
31	Total rows: 32 of 32 Query complete 00:00:03.787
32	Planning Time: 10.167 ms
	Execution Time: 3720.472 ms

Figura 13: Query Plan para la consulta 1 con índices en la tabla de 1 millón

QUERY PLAN	
text	
1	Aggregate (cost=641393.68..641393.69 rows=1 width=8) (actual time=23840.782..23840.866 rows=1 loops=1)
2	-> Finalize HashAggregate (cost=633541.74..637824.62 rows=285525 width=26) (actual time=23773.609..23825.721 rows=243082 loops=1)
3	Group Key: o.aecodigo, a.monto
4	Batches: 1 Memory Usage: 36801kB
5	-> Gather (cost=606773.72..631757.21 rows=237938 width=26) (actual time=23636.658..23685.753 rows=266248 loops=1)
6	Workers Planned: 2
7	Workers Launched: 2
8	-> Partial HashAggregate (cost=605773.72..606963.41 rows=118969 width=26) (actual time=23564.049..23583.766 rows=88749 loops=3)
9	Group Key: o.aecodigo, a.monto
10	Batches: 1 Memory Usage: 22545kB
11	Worker 0: Batches: 1 Memory Usage: 22545kB
12	Worker 1: Batches: 1 Memory Usage: 14353kB
13	-> Nested Loop (cost=1.29..604286.61 rows=118969 width=30) (actual time=18.701..23393.559 rows=92900 loops=3)
14	-> Nested Loop (cost=0.86..550590.57 rows=118969 width=40) (actual time=17.091..19911.542 rows=92900 loops=3)
15	-> Nested Loop (cost=0.43..492489.53 rows=118969 width=22) (actual time=15.514..14639.728 rows=92900 loops=3)
16	-> Parallel Seq Scan on utiliza u (cost=0.00..62500.33 rows=2083333 width=24) (actual time=0.488..334.234 rows=1666667 loops=3)
17	-> Memoize (cost=0.43..0.47 rows=1 width=18) (actual time=0.008..0.008 rows=0 loops=500000)
18	Cache Key: u.im_ns
19	Cache Mode: logical
20	Hits: 860689 Misses: 811991 Evictions: 0 Overflows: 0 Memory Usage: 61632kB
21	Worker 0: Hits: 880434 Misses: 817046 Evictions: 0 Overflows: 0 Memory Usage: 62016kB
22	Worker 1: Hits: 825511 Misses: 804329 Evictions: 0 Overflows: 0 Memory Usage: 61043kB
23	-> Index Scan using pk_insumomedicos_ns on insumo_medico i (cost=0.42..0.46 rows=1 width=18) (actual time=0.016..0.016 rows=0 loops=243366)
24	Index Cond: ((nro_serial):text = (u.im_ns):text)
25	Filter: (costo_por_unidad > 95)::double precision
26	Rows Removed by Filter: 1
27	-> Index Only Scan using pk_activideconomica_codigo on actividad_economica a (cost=0.43..0.49 rows=1 width=18) (actual time=0.056..0.056 rows=1 loops=27..)
28	Index Cond: ((codigo):text ~ (u.ocodigo):text)
29	-> Index Only Scan using pk_operacion_activideconomica_codigo on operacion o (cost=0.42..0.45 rows=1 width=10) (actual time=0.036..0.036 rows=1 loops=2..)
30	Index Cond: (aecodigo = (u.ocodigo):text)
31	Heap Fetches: 278701
32	Planning Time: 8.897 ms
33	Execution Time: 23857.599 ms
	Total rows: 33 of 33 Query complete 00:00:24.005

Figura 14: Query Plan para la consulta 1 sin índices en la tabla de 1 millón

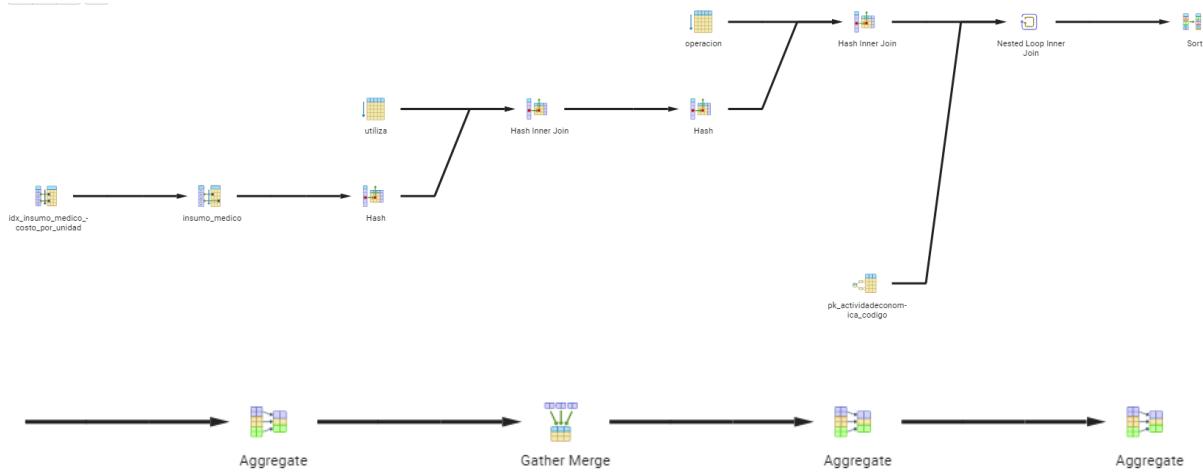


Figura 15: Query Plan Tree para la planificación con índices en la tabla de 1 millón

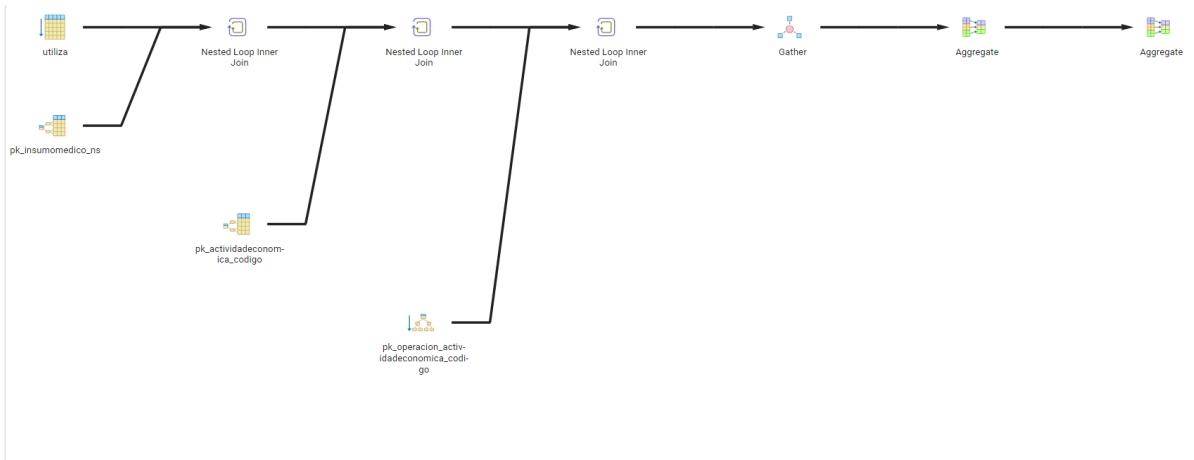


Figura 16: Query Plan Tree para la planificación sin índices en la tabla de 1 millón

5.5.2. Ejecución de consulta 2

QUERY PLAN	
	text
1	Sort (cost=294.01..294.43 rows=166 width=38) (actual time=2.613..2.623 rows=101 loops=1)
2	Sort Key: (count(o.aecodigo)) DESC
3	Sort Method: quicksort Memory: 31kB
4	-> GroupAggregate (cost=282.91..287.89 rows=166 width=38) (actual time=2.423..2.592 rows=101 loops=1)
5	Group Key: o.tipo_operacion
6	-> Sort (cost=282.91..283.32 rows=166 width=44) (actual time=2.405..2.418 rows=156 loops=1)
7	Sort Key: o.tipo_operacion, e.pdni
8	Sort Method: quicksort Memory: 36kB
9	-> Hash Join (cost=254.21..276.79 rows=166 width=44) (actual time=1.951..2.115 rows=156 loops=1)
10	Hash Cond: ((o.aecodigo)::text = (ae.codigo)::text)
11	-> Seq Scan on operacion o (cost=0.00..18.00 rows=1000 width=16) (actual time=0.009..0.062 rows=1000 loops=1)
12	-> Hash (cost=252.13..252.13 rows=166 width=48) (actual time=1.934..1.940 rows=156 loops=1)
13	Buckets: 1024 Batches: 1 Memory Usage: 22kB
14	-> Hash Join (cost=225.72..252.13 rows=166 width=48) (actual time=1.729..1.904 rows=156 loops=1)
15	Hash Cond: ((im.nro_serial)::text = (u.im_ns)::text)
16	-> Seq Scan on insumo_medico im (cost=0.00..21.00 rows=1000 width=18) (actual time=0.007..0.066 rows=1000 loops=1)
17	-> Hash (cost=223.65..223.65 rows=166 width=50) (actual time=1.716..1.721 rows=156 loops=1)
18	Buckets: 1024 Batches: 1 Memory Usage: 22kB
19	-> Hash Join (cost=113.53..223.65 rows=166 width=50) (actual time=0.723..1.680 rows=156 loops=1)
20	Hash Cond: ((u.ocodigo)::text = (ae.codigo)::text)
21	-> Hash Join (cost=17.01..122.19 rows=875 width=32) (actual time=0.163..1.037 rows=829 loops=1)
22	Hash Cond: ((u.edni)::text = (e.pdni)::text)
23	-> Seq Scan on utiliza u (cost=0.00..92.00 rows=5000 width=32) (actual time=0.007..0.286 rows=5000 loops=1)
24	-> Hash (cost=14.82..14.82 rows=175 width=8) (actual time=0.104..0.105 rows=177 loops=1)
25	Buckets: 1024 Batches: 1 Memory Usage: 16kB
26	-> Bitmap Heap Scan on enfermera e (cost=5.63..14.82 rows=175 width=8) (actual time=0.052..0.081 rows=177 loops=1)
27	Recheck Cond: (sueldo > '5500'::double precision)
28	Heap Blocks: exact=7
29	-> Bitmap Index Scan on idx_sueldo_enfermera (cost=0.00..5.59 rows=175 width=0) (actual time=0.045..0.045 rows=177 loops=1)
30	Index Cond: (sueldo > '5500'::double precision)
31	-> Hash (cost=91.77..91.77 rows=380 width=18) (actual time=0.538..0.541 rows=365 loops=1)
32	Buckets: 1024 Batches: 1 Memory Usage: 27kB
33	-> Hash Join (cost=19.50..91.77 rows=380 width=18) (actual time=0.107..0.488 rows=365 loops=1)
34	Hash Cond: ((ae.ddni)::text = (d.pdni)::text)
35	-> Seq Scan on actividad_economica ae [Close] (cost=0.00..67.00 rows=2000 width=18) (actual time=0.007..0.150 rows=2000 loops=1)
36	-> Hash (cost=17.12..17.12 rows=190 width=8) (actual time=0.095..0.096 rows=191 loops=1)
37	Buckets: 1024 Batches: 1 Memory Usage: 16kB
38	-> Bitmap Heap Scan on doctor d (cost=5.75..17.12 rows=190 width=8) (actual time=0.043..0.073 rows=191 loops=1)
39	Recheck Cond: (sueldo > '250000'::double precision)
40	Heap Blocks: exact=9
41	-> Bitmap Index Scan on idx_sueldo_doctor (cost=0.00..5.70 rows=190 width=0) (actual time=0.039..0.039 rows=191 loops=1)
42	Index Cond: (sueldo > '250000'::double precision)
43	Planning Time: 11.075 ms
44	Execution Time: 2.750 ms

Figura 17: Query Plan para la consulta 2 con índices en la tabla de 1 mil

QUERY PLAN	
text	
1	Sort (cost=20000000984.61..20000000985.03 rows=166 width=38) (actual time=12.039..12.045 rows=101 loops=1)
2	Sort Key: (count(o.aecodigo)) DESC
3	Sort Method: quicksort Memory: 31kB
4	-> GroupAggregate (cost=10000000973.51..10000000978.49 rows=166 width=38) (actual time=11.854..12.021 rows=101 loops=1)
5	Group Key: o.tipo_operacion
6	-> Sort (cost=10000000973.51..10000000973.92 rows=166 width=44) (actual time=11.835..11.843 rows=156 loops=1)
7	Sort Key: o.tipo_operacion, e.pdni
8	Sort Method: quicksort Memory: 36kB
9	-> Nested Loop (cost=1.39..967.39 rows=166 width=44) (actual time=0.378..11.537 rows=156 loops=1)
10	-> Nested Loop (cost=1.12..916.57 rows=166 width=46) (actual time=0.353..10.800 rows=156 loops=1)
11	-> Nested Loop (cost=0.84..626.47 rows=950 width=46) (actual time=0.216..7.281 rows=925 loops=1)
12	Join Filter: ((o.aecodigo)::text = (u.ocodigo)::text)
13	-> Nested Loop (cost=0.56..515.80 rows=190 width=34) (actual time=0.200..5.792 rows=184 loops=1)
14	-> Nested Loop (cost=0.29..394.01 rows=380 width=18) (actual time=0.165..4.344 rows=365 loops=1)
15	-> Seq Scan on actividad_economica ae (cost=0.00..67.00 rows=2000 width=18) (actual time=0.021..0.195 rows=2000 loops=1)
16	-> Memoize (cost=0.29..0.33 rows=1 width=8) (actual time=0.002..0.002 rows=0 loops=2000)
17	Cache Key: ae.ddni
18	Cache Mode: logical
19	Hits: 1148 Misses: 852 Evictions: 0 Overflows: 0 Memory Usage: 67kB
20	-> Index Scan using pk_doctor_pdni on doctor d (cost=0.28..0.32 rows=1 width=8) (actual time=0.004..0.004 rows=0 loops=852)
21	Index Cond: ((pdni)::text = (ae.ddni)::text)
22	Filter: (sueldo > '250000)::double precision
23	Rows Removed by Filter: 1
24	-> Index Scan using pk_operacion_actividadeconomico_codigo on operacion o (cost=0.28..0.32 rows=1 width=16) (actual time=0.004..0.004 rows=1 loops=1)
25	Index Cond: ((aecodigo)::text = (ae.codigo)::text)
26	-> Index Scan using pk_utiliza_ocodigoe_dnini_imns on utiliza u (cost=0.28..0.52 rows=5 width=32) (actual time=0.005..0.007 rows=5 loops=184)
27	Index Cond: ((ocodigo)::text = (ae.codigo)::text)
28	-> Index Scan using pk_enfermera_pdni on enfermera e (cost=0.28..0.31 rows=1 width=8) (actual time=0.004..0.004 rows=0 loops=925)
29	Index Cond: ((pdni)::text = (u.edni)::text)
30	Filter: (sueldo > '5500)::double precision
31	Rows Removed by Filter: 1
32	-> Index Scan using pk_insumomedico_ns on insumo_medico im (cost=0.28..0.31 rows=1 width=18) (actual time=0.004..0.004 rows=1 loops=156)
33	Index Cond: ((nro_serial)::text = (u.im_ns)::text)
34	Planning Time: 8.495 ms
35	Execution Time: 12.142 ms
Total rows: 35 of 35 Query complete 00:00:00.001	

Figura 18: Query Plan para la consulta 2 sin índices en la tabla de 1 mil

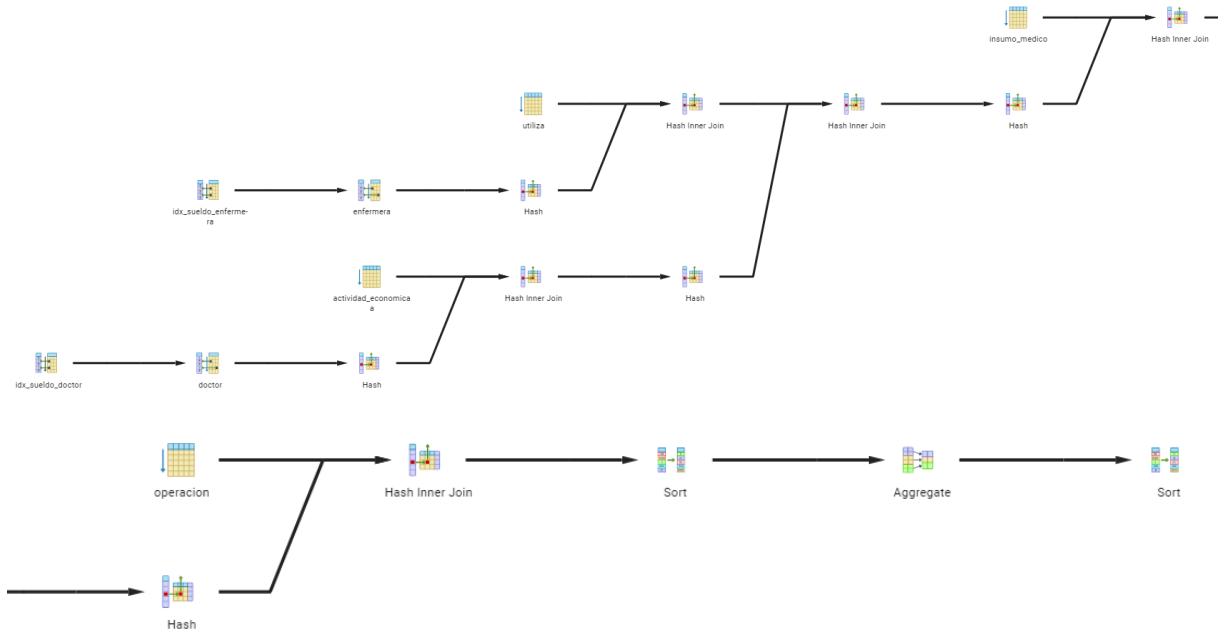


Figura 19: Query Plan Tree para la planificación con índices en la tabla de 1 mil

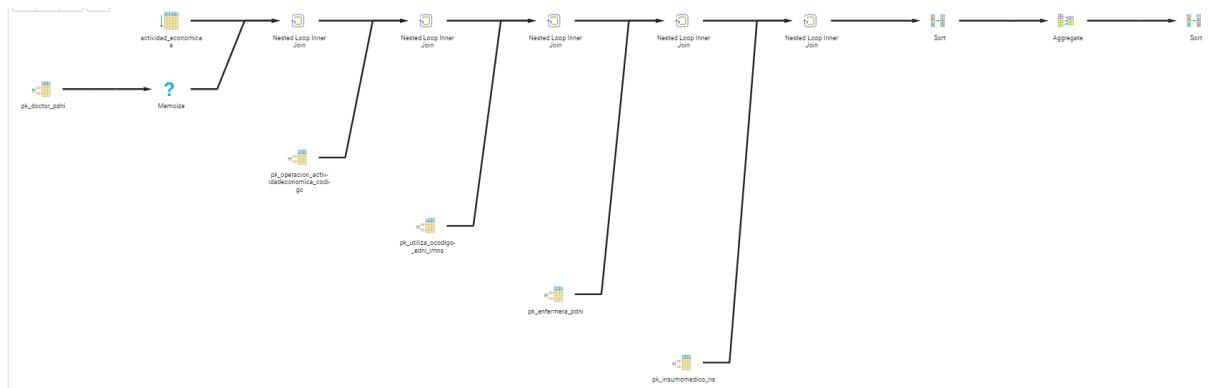


Figura 20: Query Plan Tree para la planificación sin índices en la tabla de 1 mil

QUERY PLAN	
text	
1	Sort (cost=2858.85..2861.28 rows=970 width=38) (actual time=28.215..28.241 rows=597 loops=1)
2	Sort Key: (count(o.aecodigo)) DESC
3	Sort Method: quicksort Memory: 63kB
4	-> GroupAggregate (cost=2772.07..2810.73 rows=970 width=38) (actual time=26.814..28.123 rows=597 loops=1)
5	Group Key: o.tipo_operacion
6	-> Sort (cost=2772.07..2775.69 rows=1448 width=44) (actual time=26.728..26.791 rows=1372 loops=1)
7	Sort Key: o.tipo_operacion, e.pdni
8	Sort Method: quicksort Memory: 152kB
9	-> Hash Join (cost=2434.07..2696.05 rows=1448 width=44) (actual time=21.210..22.978 rows=1372 loops=1)
10	Hash Cond: ((im.nro_serial)::text = (u.im_ns)::text)
11	-> Seq Scan on insumo_medico im (cost=0.00..210.00 rows=10000 width=18) (actual time=0.009..0.615 rows=10000 loops=1)
12	-> Hash (cost=2415.97..2415.97 rows=1448 width=46) (actual time=21.187..21.193 rows=1372 loops=1)
13	Buckets: 2048 Batches: 1 Memory Usage: 128kB
14	-> Hash Join (cost=1323.15..2415.97 rows=1448 width=46) (actual time=9.793..20.826 rows=1372 loops=1)
15	Hash Cond: ((u.ocodigo)::text = (o.aecodigo)::text)
16	-> Hash Join (cost=135.75..1184.06 rows=8010 width=32) (actual time=0.547..10.418 rows=8053 loops=1)
17	Hash Cond: ((u.edni)::text = (e.pdni)::text)
18	-> Seq Scan on utiliza u (cost=0.00..917.00 rows=50000 width=32) (actual time=0.008..3.108 rows=50000 loops=1)
19	-> Hash (cost=115.73..115.73 rows=1602 width=8) (actual time=0.530..0.531 rows=1602 loops=1)
20	Buckets: 2048 Batches: 1 Memory Usage: 80kB
21	-> Bitmap Heap Scan on enfermera e (cost=32.70..115.73 rows=1602 width=8) (actual time=0.116..0.365 rows=1602 loops=1)
22	Recheck Cond: (sueldo > '5500'::double precision)
23	Heap Blocks: exact=63
24	-> Bitmap Index Scan on idx_sueldo_enfermera (cost=0.00..32.30 rows=1602 width=0) (actual time=0.107..0.107 rows=1602 loops=1)
25	Index Cond: (sueldo > '5500'::double precision)
26	-> Hash (cost=1164.80..1164.80 rows=1808 width=34) (actual time=9.232..9.234 rows=1728 loops=1)
27	Buckets: 2048 Batches: 1 Memory Usage: 133kB
28	-> Hash Join (cost=936.22..1164.80 rows=1808 width=34) (actual time=5.848..8.781 rows=1728 loops=1)
29	Hash Cond: ((o.aecodigo)::text = (ae.codigo)::text)
30	-> Seq Scan on operacion o (cost=0.00..173.00 rows=10000 width=16) (actual time=0.011..0.883 rows=10000 loops=1)
31	-> Hash (cost=891.02..891.02 rows=3616 width=18) (actual time=5.821..5.823 rows=3565 loops=1)
32	Buckets: 4096 Batches: 1 Memory Usage: 213kB
33	-> Hash Join (cost=172.50..891.02 rows=3616 width=18) (actual time=0.681..5.180 rows=3565 loops=1)
34	Hash Cond: ((ae.ddni)::text = (d.pdni)::text)
35	-> Seq Scan on actividad_economica ae (cost=0.00..666.00 rows=20000 width=18) (actual time=0.006..1.696 rows=20000 loops=1)
36	-> Hash (cost=149.90..149.90 rows=1808 width=8) (actual time=0.668..0.669 rows=1806 loops=1)
37	Buckets: 2048 Batches: 1 Memory Usage: 89kB
38	-> Bitmap Heap Scan on doctor d (cost=38.30..149.90 rows=1808 width=8) (actual time=0.112..0.458 rows=1806 loops=1)
39	Recheck Cond: (sueldo > '250000'::double precision)
40	Heap Blocks: exact=89
41	-> Bitmap Index Scan on idx_sueldo_doctor (cost=0.00..37.84 rows=1808 width=0) (actual time=0.102..0.102 rows=1806 loops=1)
42	Index Cond: (sueldo > '250000'::double precision)
43	Planning Time: 9.744 ms
44	Execution Time: 28.466 ms
Total rows: 44 of 44 Query complete 00:00:00.103	

Figura 21: Query Plan para la consulta 2 con índices en la tabla de 10 mil

QUERY PLAN	
text	
1	Sort (cost=20000009931.46..20000009933.88 rows=970 width=38) (actual time=292.298..292.348 rows=597 loops=1)
2	Sort Key: (count(o.aecodigo)) DESC
3	Sort Method: quicksort Memory: 63kB
4	-> GroupAggregate (cost=10000009844.67..10000009883.33 rows=970 width=38) (actual time=289.912..292.117 rows=597 loops=1)
5	Group Key: o.tipo_operacion
6	-> Sort (cost=10000009844.67..10000009848.29 rows=1448 width=44) (actual time=289.864..290.042 rows=1372 loops=1)
7	Sort Key: o.tipo_operacion, e.pdni
8	Sort Method: quicksort Memory: 152kB
9	-> Nested Loop (cost=1.57..9768.66 rows=1448 width=44) (actual time=0.758..285.076 rows=1372 loops=1)
10	-> Nested Loop (cost=1.28..9313.14 rows=1448 width=46) (actual time=0.732..274.985 rows=1372 loops=1)
11	-> Nested Loop (cost=1.00..6481.48 rows=9040 width=46) (actual time=0.521..220.781 rows=8631 loops=1)
12	Join Filter: ((o.aecodigo)::text = (u.ocodigo)::text)
13	-> Nested Loop (cost=0.58..5198.53 rows=1808 width=34) (actual time=0.127..85.905 rows=1728 loops=1)
14	-> Nested Loop (cost=0.30..4022.24 rows=3616 width=18) (actual time=0.102..65.148 rows=3565 loops=1)
15	-> Seq Scan on actividad_economica ae (cost=0.00..666.00 rows=20000 width=18) (actual time=0.032..2.359 rows=20000 loops=1)
16	-> Memoize (cost=0.30..0.34 rows=1 width=8) (actual time=0.003..0.003 rows=0 loops=20000)
17	Cache Key: ae.ddni
18	Cache Mode: logical
19	Hits: 11366 Misses: 8634 Evictions: 0 Overflows: 0 Memory Usage: 677kB
20	-> Index Scan using pk_doctor_pdni on doctor d (cost=0.29..0.33 rows=1 width=8) (actual time=0.006..0.006 rows=0 loops=8634)
21	Index Cond: ((pdni)::text = (ae.ddni)::text)
22	Filter: (sueldo > '250000'::double precision)
23	Rows Removed by Filter: 1
24	-> Index Scan using pk_operacion_actividadadeconomica_codigo on operacion o (cost=0.29..0.33 rows=1 width=16) (actual time=0.005..0.005 rows=0 loops=...)
25	Index Cond: ((aecodigo)::text = (ae.codigo)::text)
26	-> Index Scan using pk_utiliza_ocodigo_edni_imns on utiliza u (cost=0.41..0.65 rows=5 width=32) (actual time=0.042..0.076 rows=5 loops=1728)
27	Index Cond: ((ocodigo)::text = (ae.codigo)::text)
28	-> Index Scan using pk_enfermera_pdni on enfermera e (cost=0.29..0.31 rows=1 width=8) (actual time=0.006..0.006 rows=0 loops=8631)
29	Index Cond: ((pdni)::text = (u.edni)::text)
30	Filter: (sueldo > '5500'::double precision)
31	Rows Removed by Filter: 1
32	-> Index Scan using pk_insumomedico_ns on insumo_medico im (cost=0.29..0.31 rows=1 width=18) (actual time=0.007..0.007 rows=1 loops=1372)
33	Index Cond: ((nro_serial)::text = (u.im_ns)::text)
34	Planning Time: 9.050 ms
35	Execution Time: 292.691 ms
Total rows: 35 of 35 Query complete 00:00:00 345	

Figura 22: Query Plan para la consulta 2 sin índices en la tabla de 10 mil

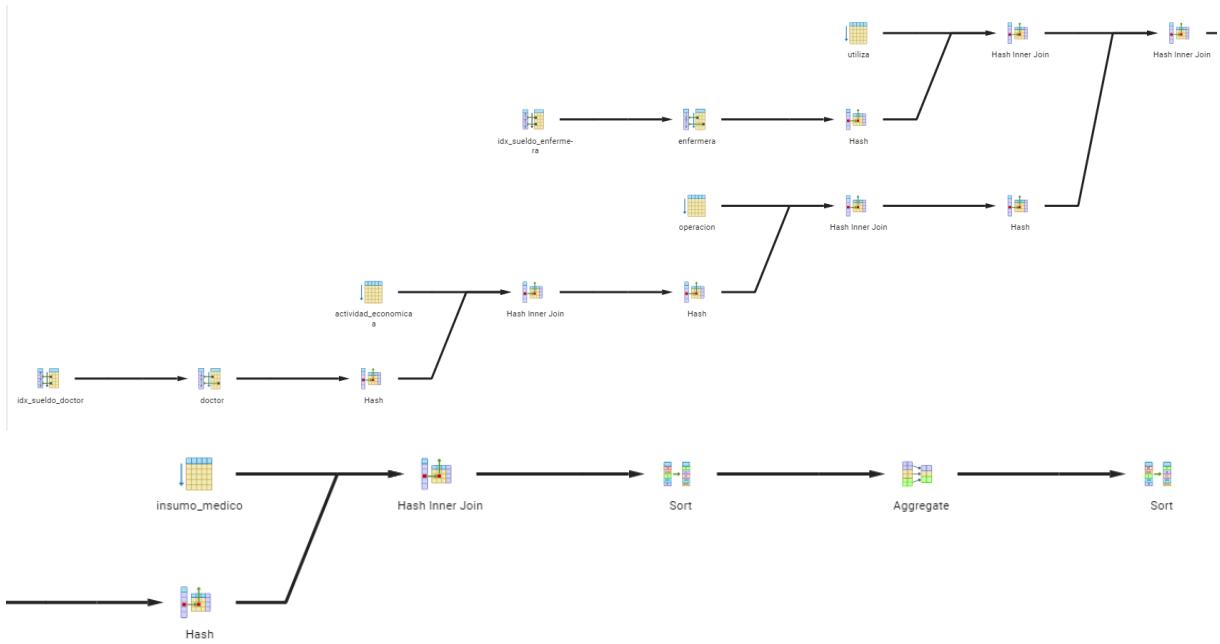


Figura 23: Query Plan Tree para la planificación con índices en la tabla de 10 mil

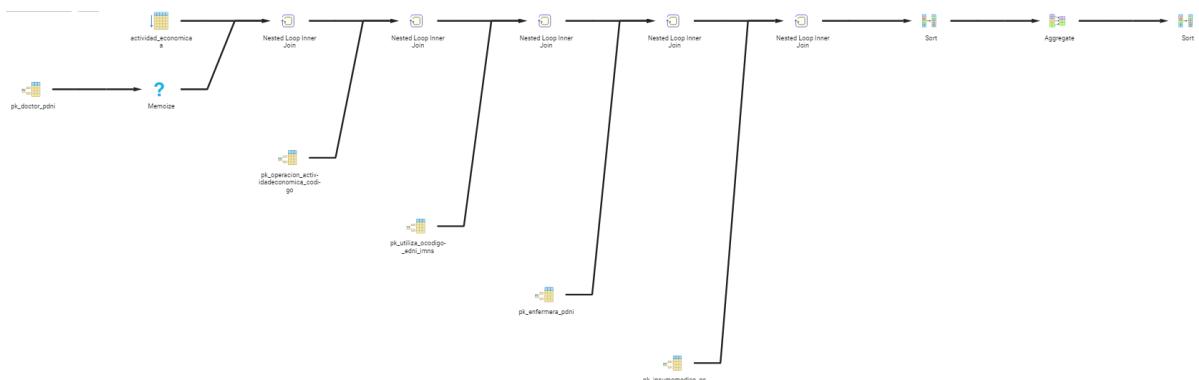


Figura 24: Query Plan Tree para la planificación sin índices en la tabla de 10 mil

QUERY PLAN	
text	
1	Sort (cost=23462.56..23464.99 rows=971 width=38) (actual time=432.852..439.615 rows=971 loops=1)
2	Sort Key: (count(o.aecodigo)) DESC
3	Sort Method: quicksort Memory: 86kB
4	-> GroupAggregate (cost=21326.12..23414.38 rows=971 width=38) (actual time=406.596..439.300 rows=971 loops=1)
5	Group Key: o.tipo_operacion
6	-> Gather Merge (cost=21326.12..23128.00 rows=15810 width=44) (actual time=406.543..421.155 rows=15750 loops=1)
7	Workers Planned: 1
8	Workers Launched: 1
9	-> Sort (cost=20326.11..20349.36 rows=9300 width=44) (actual time=322.610..323.657 rows=7875 loops=2)
10	Sort Key: o.tipo_operacion, e.pdni
11	Sort Method: quicksort Memory: 803kB
12	Worker 0: Sort Method: quicksort Memory: 765kB
13	-> Parallel Hash Join (cost=17767.53..19713.10 rows=9300 width=44) (actual time=277.259..291.299 rows=7875 loops=2)
14	Hash Cond: ((im.nro_serial)::text = (u.im_ns)::text)
15	-> Parallel Seq Scan on insumo_medico im (cost=0.00..1686.24 rows=58824 width=18) (actual time=0.007..3.991 rows=50000 loops=2)
16	-> Parallel Hash (cost=17685.18..17685.18 rows=6588 width=46) (actual time=276.038..276.045 rows=7875 loops=2)
17	Buckets: 16384 Batches: 1 Memory Usage: 1504kB
18	-> Hash Join (cost=6052.03..17685.18 rows=6588 width=46) (actual time=50.125..271.931 rows=7875 loops=2)
19	Hash Cond: ((u.edni)::text = (e.pdni)::text)
20	-> Nested Loop (cost=4674.06..16205.46 rows=38762 width=46) (actual time=42.348..249.326 rows=46872 loops=2)
21	Join Filter: ((o.aecodigo)::text = (u.ocodigo)::text)
22	-> Hash Join (cost=4673.64..10418.44 rows=7752 width=34) (actual time=42.299..106.079 rows=9398 loops=2)
23	Hash Cond: ((ae.codigo)::text = (o.aecodigo)::text)
24	-> Hash Join (cost=1702.64..7406.74 rows=15505 width=18) (actual time=10.202..65.929 rows=18740 loops=2)
25	Hash Cond: ((ae.ddni)::text = (d.pdni)::text)
26	-> Parallel Seq Scan on actividad_economica ae (cost=0.00..5485.34 rows=83334 width=18) (actual time=0.651..27.338 rows=100000 ...)
27	-> Hash (cost=1470.06..1470.06 rows=18606 width=8) (actual time=9.474..9.475 rows=18696 loops=2)
28	Buckets: 32768 Batches: 1 Memory Usage: 1003kB
29	-> Bitmap Heap Scan on doctor d (cost=352.49..1470.06 rows=18606 width=8) (actual time=1.306..5.984 rows=18696 loops=2)
30	Recheck Cond: (sueldo > '250000)::double precision
31	Heap Blocks: exact=885
32	-> Bitmap Index Scan on idx_sueldo_doctor (cost=0.00..347.84 rows=18606 width=0) (actual time=1.210..1.210 rows=18696 loops=1)
33	Index Cond: (sueldo > '250000)::double precision
34	-> Hash (cost=1721.00..1721.00 rows=100000 width=16) (actual time=31.907..31.907 rows=100000 loops=2)
35	Buckets: 131072 Batches: 1 Memory Usage: 5852kB
36	-> Seq Scan on operacion o (cost=0.00..1721.00 rows=100000 width=16) (actual time=0.402..11.363 rows=100000 loops=2)
37	-> Index Scan using pk_utiliza_ocodigo_edni_imns on utiliza u (cost=0.42..0.67 rows=6 width=32) (actual time=0.010..0.014 rows=5 loops=18...)
38	Index Cond: ((ocodigo)::text = (ae.codigo)::text)
39	-> Hash (cost=1165.55..1165.55 rows=16994 width=8) (actual time=7.695..7.696 rows=16809 loops=2)
40	Buckets: 32768 Batches: 1 Memory Usage: 928kB
41	-> Bitmap Heap Scan on enfermera e (cost=324.12..1165.55 rows=16994 width=8) (actual time=1.128..5.013 rows=16809 loops=2)
42	Recheck Cond: (sueldo > '5500)::double precision
43	Heap Blocks: exact=629
44	-> Bitmap Index Scan on idx_sueldo_enfermera (cost=0.00..319.87 rows=16994 width=0) (actual time=1.054..1.054 rows=16809 loops=2)
45	Index Cond: (sueldo > '5500)::double precision
46	Planning Time: 5.747 ms
47	Execution Time: 440.765 ms
Total rows: 47 of 47 Query complete 00:00:00.509	

Figura 25: Query Plan para la consulta 2 con índices en la tabla de 100 mil

QUERY PLAN	
text	
1	Sort (cost=20000076718.65..20000076721.08 rows=971 width=38) (actual time=1217.362..1223.813 rows=971 loops=1)
2	Sort Key: (count(o.aecodigo)) DESC
3	Sort Method: quicksort Memory: 86kB
4	-> GroupAggregate (cost=10000074542.75..10000076670.48 rows=971 width=38) (actual time=1182.934..1223.413 rows=971 loops=1)
5	Group Key: o.tipo_operacion
6	-> Gather Merge (cost=10000074542.75..10000076384.09 rows=15810 width=44) (actual time=1182.884..1203.227 rows=15750 loops=1)
7	Workers Planned: 2
8	Workers Launched: 2
9	-> Sort (cost=10000073542.73..10000073559.20 rows=6588 width=44) (actual time=1151.234..1151.849 rows=5250 loops=3)
10	Sort Key: o.tipo_operacion, e.pdni
11	Sort Method: quicksort Memory: 611kB
12	Worker 0: Sort Method: quicksort Memory: 579kB
13	Worker 1: Sort Method: quicksort Memory: 571kB
14	-> Nested Loop (cost=2.10..73124.87 rows=6588 width=44) (actual time=1.793..1130.617 rows=5250 loops=3)
15	-> Nested Loop (cost=1.69..70180.77 rows=6588 width=46) (actual time=1.360..1073.775 rows=5250 loops=3)
16	-> Nested Loop (cost=1.27..52910.75 rows=38762 width=46) (actual time=1.311..775.773 rows=31248 loops=3)
17	Join Filter: ((o.aecodigo)::text = (u.ocodigo)::text)
18	-> Nested Loop (cost=0.84..47123.73 rows=7752 width=34) (actual time=1.271..662.023 rows=6266 loops=3)
19	-> Nested Loop (cost=0.43..40035.16 rows=15505 width=18) (actual time=0.759..543.733 rows=12493 loops=3)
20	-> Parallel Seq Scan on actividad_economica ae (cost=0.00..5485.34 rows=83334 width=18) (actual time=0.565..39.128 rows=66667 loops=3)
21	-> Memoize (cost=0.43..0.47 rows=1 width=8) (actual time=0.007..0.007 rows=0 loops=200001)
22	Cache Key: ae.ddni
23	Cache Mode: logical
24	Hits: 18428 Misses: 48824 Evictions: 0 Overflows: 0 Memory Usage: 3839kB
25	Worker 0: Hits: 19492 Misses: 50082 Evictions: 0 Overflows: 0 Memory Usage: 3939kB
26	Worker 1: Hits: 16558 Misses: 46617 Evictions: 0 Overflows: 0 Memory Usage: 3667kB
27	-> Index Scan using pk_doctor_pdni on doctor d (cost=0.42..0.46 rows=1 width=8) (actual time=0.009..0.009 rows=0 loops=145523)
28	Index Cond: ((pdni)::text = (ae.ddni)::text)
29	Filter: (sueldo > '250000)::double precision
30	Rows Removed by Filter: 1
R	--
R	31 -> Index Scan using pk_operacion_actividad_economica_codigo on operacion o (cost=0.42..0.46 rows=1 width=16) (actual time=0.009..0.009 rows=1 loops=3...)
T	32 Index Cond: ((aecodigo)::text = (ae.codigo)::text)
ei	33 -> Index Scan using pk_utiliza_ocodigo_edni_imns on utiliza u (cost=0.42..0.67 rows=6 width=32) (actual time=0.012..0.016 rows=5 loops=18797)
ei	34 Index Cond: ((ocodigo)::text = (ae.codigo)::text)
ci	35 -> Index Scan using pk_enfermera_pdni on enfermera e (cost=0.42..0.45 rows=1 width=8) (actual time=0.009..0.009 rows=0 loops=93744)
g	36 Index Cond: ((pdni)::text = (u.edni)::text)
C	37 Filter: (sueldo > '5500)::double precision
C	38 Rows Removed by Filter: 1
Ir	39 -> Index Scan using pk_insumo_medico_ns on insumo_medico im (cost=0.42..0.45 rows=1 width=18) (actual time=0.010..0.010 rows=1 loops=15750)
R	40 Index Cond: ((nro_serial)::text = (u.im_ns)::text)
R	41 Planning Time: 8.948 ms
T	42 Execution Time: 1225.124 ms
di	Total rows: 42 of 42 Query complete 00:00:01.269

Figura 26: Query Plan para la consulta 2 sin índices en la tabla de 100 mil

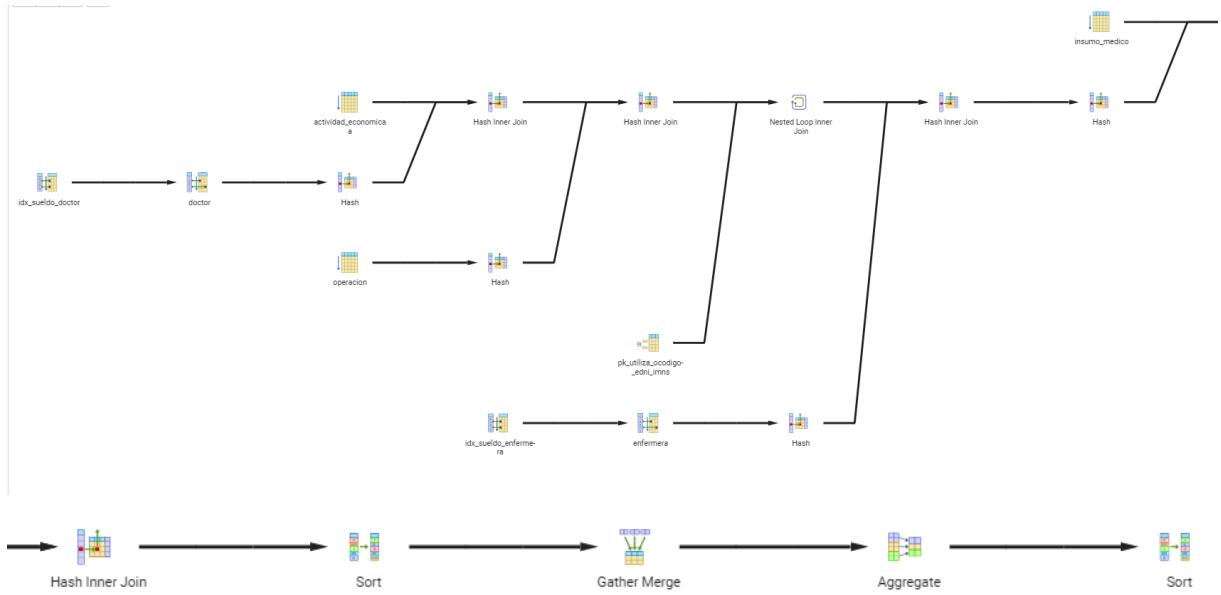


Figura 27: Query Plan Tree para la planificación con índices en la tabla de 100 mil

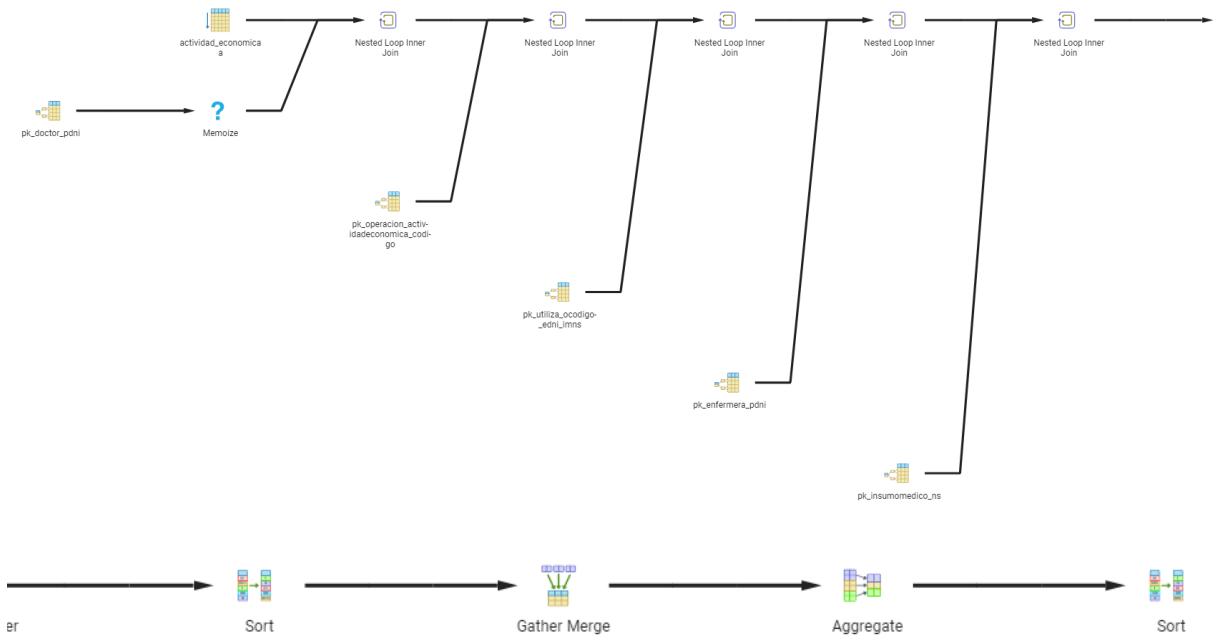


Figura 28: Query Plan Tree para la planificación sin índices en la tabla de 100 mil

QUERY PLAN	
text	
1	Sort (cost=220235.71..220238.14 rows=971 width=38) (actual time=9249.270..9288.024 rows=971 loops=1)
2	Sort Key: (count(o.aecodigo)) DESC
3	Sort Method: quicksort Memory: 86kB
4	-> GroupAggregate (cost=199434.45..220187.53 rows=971 width=38) (actual time=8714.346..9287.009 rows=971 loops=1)
5	Group Key: o.tipo_operacion
6	-> Gather Merge (cost=199434.45..217468.12 rows=154840 width=44) (actual time=8713.670..8934.442 rows=153757 loops=1)
7	Workers Planned: 2
8	Workers Launched: 2
9	-> Sort (cost=198434.43..198595.72 rows=64517 width=44) (actual time=8663.900..8696.768 rows=51252 loops=3)
10	Sort Key: o.tipo_operacion, e.pdni
11	Sort Method: external merge Disk: 3032kB
12	Worker 0: Sort Method: external merge Disk: 2928kB
13	Worker 1: Sort Method: external merge Disk: 3192kB
14	-> Parallel Hash Join (cost=174320.37..191292.36 rows=64517 width=44) (actual time=8147.150..8339.208 rows=51252 loops=3)
15	Hash Cond: ((im.nro_serial)::text = (u.im_ns)::text)
16	-> Parallel Seq Scan on insumo_medico im (cost=0.00..15140.67 rows=416667 width=18) (actual time=0.698..65.385 rows=333333 loops=3)
17	-> Parallel Hash (cost=173513.91..173513.91 rows=64517 width=46) (actual time=8145.619..8146.118 rows=51252 loops=3)
18	Buckets: 262144 Batches: 1 Memory Usage: 15296kB
19	-> Parallel Hash Join (cost=85330.78..173513.91 rows=64517 width=46) (actual time=769.610..8084.231 rows=51252 loops=3)
20	Hash Cond: ((u.edni)::text = (e.pdni)::text)
21	-> Nested Loop (cost=74229.04..161386.04 rows=390904 width=46) (actual time=725.023..7813.890 rows=308256 loops=3)
22	Join Filter: ((o.aecodigo)::text = (u.ocodigo)::text)
23	-> Parallel Hash Join (cost=74228.48..92473.40 rows=78181 width=34) (actual time=723.815..959.860 rows=61620 loops=3)
24	Hash Cond: ((o.aecodigo)::text = (ae.codigo)::text)
25	-> Parallel Seq Scan on operacion o (cost=0.00..11369.67 rows=416667 width=16) (actual time=0.891..65.753 rows=333333 loops=3)
26	-> Parallel Hash (cost=71356.96..71356.96 rows=156362 width=18) (actual time=554.384..554.873 rows=123093 loops=3)
27	Buckets: 131072 Batches: 4 Memory Usage: 6144kB
28	-> Parallel Hash Join (cost=14324.11..71356.96 rows=156362 width=18) (actual time=78.774..514.948 rows=123093 loops=3)
29	Hash Cond: ((ae.ddni)::text = (d.pdn)::text)
30	-> Parallel Seq Scan on actividad_economica ae (cost=0.00..54845.33 rows=833333 width=18) (actual time=0.882..199.810 rows=666667 lo...
31	-> Parallel Hash (cost=13346.85..13346.85 rows=78181 width=8) (actual time=77.644..77.646 rows=61673 loops=3)
32	Buckets: 262144 Batches: 1 Memory Usage: 10656kB
33	-> Parallel Bitmap Heap Scan on doctor d (cost=3518.59..13346.85 rows=78181 width=8) (actual time=13.940..60.571 rows=61673 loops=3)
34	Recheck Cond: (sueldo > '250000)::double precision
35	Heap Blocks: exact=3259
36	-> Bitmap Index Scan on idx_sueldo_doctor (cost=0.00..3471.68 rows=187634 width=0) (actual time=12.922..12.923 rows=185018 loops=1)
37	Index Cond: (sueldo > '250000)::double precision
38	-> Index Scan using pk_utiliza_ocodigo_edni_imns on utiliza u (cost=0.56..0.81 rows=6 width=32) (actual time=0.046..0.108 rows=5 loops=184861)
39	Index Cond: ((ocodigo)::text = (ae.codigo)::text)
40	-> Parallel Hash (cost=10242.13..10242.13 rows=68769 width=8) (actual time=44.292..44.294 rows=55577 loops=3)
41	Buckets: 262144 Batches: 1 Memory Usage: 9824kB
42	-> Parallel Bitmap Heap Scan on enfermera e (cost=3095.52..10242.13 rows=68769 width=8) (actual time=4.384..30.412 rows=55577 loops=3)
43	Recheck Cond: (sueldo > '5500)::double precision
44	Heap Blocks: exact=3972
45	-> Bitmap Index Scan on idx_sueldo_enfermera (cost=0.00..3054.26 rows=165045 width=0) (actual time=12.301..12.301 rows=166730 loops=1)
46	Index Cond: (sueldo > '5500)::double precision
47	Planning Time: 9.881 ms
48	Execution Time: 9290.042 ms
Total rows: 48 of 48 Query complete 00:00:09.362	

Figura 29: Query Plan para la consulta 2 con índices en la tabla de 1 millón

QUERY PLAN	
text	
1	Sort (cost=20000787310.49..20000787312.91 rows=971 width=38) (actual time=34758.891..34763.675 rows=971 loops=1)
2	Sort Key: (count(o.aecodigo)) DESC
3	Sort Method: quicksort Memory: 86kB
4	-> GroupAggregate (cost=10000766509.23..10000787262.31 rows=971 width=38) (actual time=34293.298..34763.187 rows=971 loops=1)
5	Group Key: o.tipo_operacion
6	-> Gather Merge (cost=10000766509.23..10000784542.90 rows=154840 width=44) (actual time=34292.599..34453.670 rows=153757 loops=1)
7	Workers Planned: 2
8	Workers Launched: 2
9	-> Sort (cost=10000765509.20..10000765670.50 rows=64517 width=44) (actual time=34219.904..34250.718 rows=51252 loops=3)
10	Sort Key: o.tipo_operacion, e.pdni
11	Sort Method: external merge Disk: 3096kB
12	Worker 0: Sort Method: external merge Disk: 3072kB
13	Worker 1: Sort Method: external merge Disk: 2984kB
14	-> Nested Loop (cost=2.26..758367.14 rows=64517 width=44) (actual time=4.024..33915.710 rows=51252 loops=3)
15	-> Nested Loop (cost=1.84..729053.14 rows=64517 width=46) (actual time=3.436..31902.504 rows=51252 loops=3)
16	-> Nested Loop (cost=1.41..551965.19 rows=390904 width=46) (actual time=2.451..23897.771 rows=308256 loops=3)
17	Join Filter: ((o.aecodigo)::text = (u.ocodigo)::text)
18	-> Nested Loop (cost=0.85..483052.54 rows=78181 width=34) (actual time=1.586..15965.316 rows=61620 loops=3)
19	-> Nested Loop (cost=0.43..287143.22 rows=416667 width=34) (actual time=1.056..7245.757 rows=333333 loops=3)
20	-> Parallel Seq Scan on operacion o (cost=0.00..11369.67 rows=416667 width=16) (actual time=0.295..94.325 rows=333333 loops=3)
21	-> Index Scan using pk_actividadeconomico_codigo on actividad_economica ae (cost=0.43..0.66 rows=1 width=18) (actual time=0.021..0.021 rows=1 loops=100...)
22	Index Cond: ((codigo)::text = (o.aecodigo)::text)
23	-> Index Scan using pk_doctor_pdni on doctor d (cost=0.42..0.47 rows=1 width=8) (actual time=0.026..0.026 rows=0 loops=1000000)
24	Index Cond: ((pdni)::text = (ae.ddni)::text)
25	Filter: (sueldo > 250000::double precision)
26	Rows Removed by Filter: 1
27	-> Index Scan using pk_utiliza_ocodigo_edni_imns on utiliza u (cost=0.56..0.81 rows=6 width=32) (actual time=0.067..0.125 rows=5 loops=184861)
28	Index Cond: ((ocodigo)::text = (ae.codigo)::text)
29	-> Index Scan using pk_enfermera_pdni on enfermera e (cost=0.42..0.45 rows=1 width=8) (actual time=0.025..0.025 rows=0 loops=924767)
30	Index Cond: ((pdni)::text = (u.edni)::text)
31	Index Cond: ((pdni)::text = (u.edni)::text)
32	Filter: (sueldo > '5500)::double precision)
33	Rows Removed by Filter: 1
34	-> Index Scan using pk_insumomedico_ns on insumo_medico im (cost=0.42..0.45 rows=1 width=18) (actual time=0.038..0.038 rows=1 loops=153757)
35	Index Cond: ((nro_serial)::text = (u.im_ns)::text)
36	Planning Time: 1.328 ms
	Execution Time: 34765.752 ms
	Total rows: 36 of 36 Query complete 00:00:35.397

Figura 30: Query Plan para la consulta 2 sin índices en la tabla de 1 millón

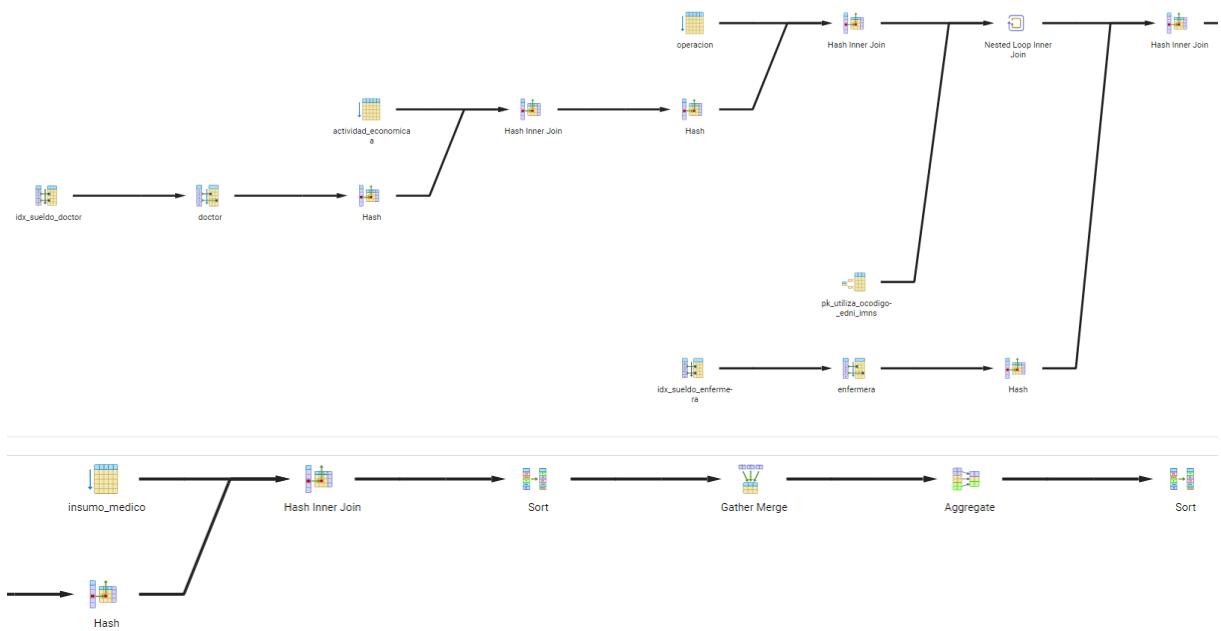


Figura 31: Query Plan Tree para la planificación con índices en la tabla de 1 millón

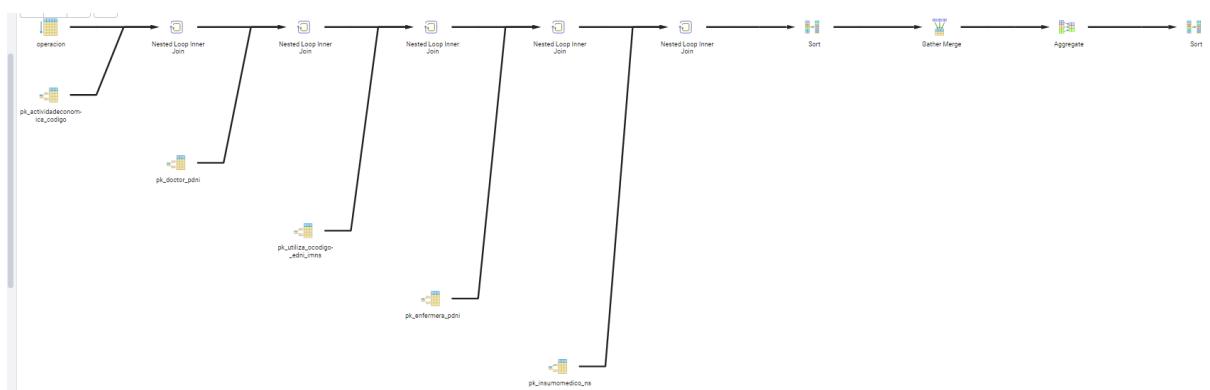


Figura 32: Query Plan Tree para la planificación sin índices en la tabla de 1 millón

5.5.3. Ejecucion de consulta 3

	QUERY PLAN	text
1	HashAggregate	(cost=2863.44..2913.64 rows=5020 width=153) (actual time=51.722..54.025 rows=5020 loops=1)
2	Group Key:	d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
3	Batches:	1 Memory Usage: 1745kB
4	-> Hash Join	(cost=1553.64..2788.14 rows=5020 width=153) (actual time=16.130..47.403 rows=5020 loops=1)
5	Hash Cond:	((p.dni)::text = (ae.ddni)::text)
6	-> Hash Join	(cost=314.00..1489.91 rows=10000 width=48) (actual time=3.671..30.486 rows=10000 loops=1)
7	Hash Cond:	((p.dni)::text = (d.pdni)::text)
8	-> Seq Scan	on persona p (cost=0.00..1022.27 rows=58527 width=22) (actual time=0.011..10.021 rows=60000 loops=1)
9	-> Hash	(cost=189.00..189.00 rows=10000 width=26) (actual time=3.624..3.628 rows=10000 loops=1)
10	Buckets:	16384 Batches: 1 Memory Usage: 710kB
11	-> Seq Scan	on doctor d (cost=0.00..189.00 rows=10000 width=26) (actual time=0.010..1.560 rows=10000 loops=1)
12	-> Hash	(cost=1176.89..1176.89 rows=5020 width=121) (actual time=12.429..12.434 rows=5020 loops=1)
13	Buckets:	8192 Batches: 1 Memory Usage: 829kB
14	-> Hash Join	(cost=385.69..1176.89 rows=5020 width=121) (actual time=2.789..10.624 rows=5020 loops=1)
15	Hash Cond:	((ae.codigo)::text = (pg.aecodigo)::text)
16	-> Seq Scan	on actividad_economica ae (cost=0.00..666.00 rows=20000 width=119) (actual time=0.019..2.190 rows=20000 loops=1)
17	-> Hash	(cost=322.94..322.94 rows=5020 width=22) (actual time=2.741..2.743 rows=5020 loops=1)
18	Buckets:	8192 Batches: 1 Memory Usage: 334kB
19	-> Bitmap Heap Scan	on pago pg (cost=63.19..322.94 rows=5020 width=22) (actual time=0.251..1.604 rows=5020 loops=1)
20	Recheck Cond:	((tipo_de_pago)::text = 'Efectivo'::text)
21	Heap Blocks:	exact=197
22	-> Bitmap Index Scan	on idx_tipo_de_pago (cost=0.00..61.94 rows=5020 width=0) (actual time=0.219..0.220 rows=5020 loops=1)
23	Index Cond:	((tipo_de_pago)::text = 'Efectivo'::text)
24	Planning Time:	6.986 ms
25	Execution Time:	54.933 ms

Figura 33: Query Plan para la consulta 3 con índices en la tabla de 10 mil

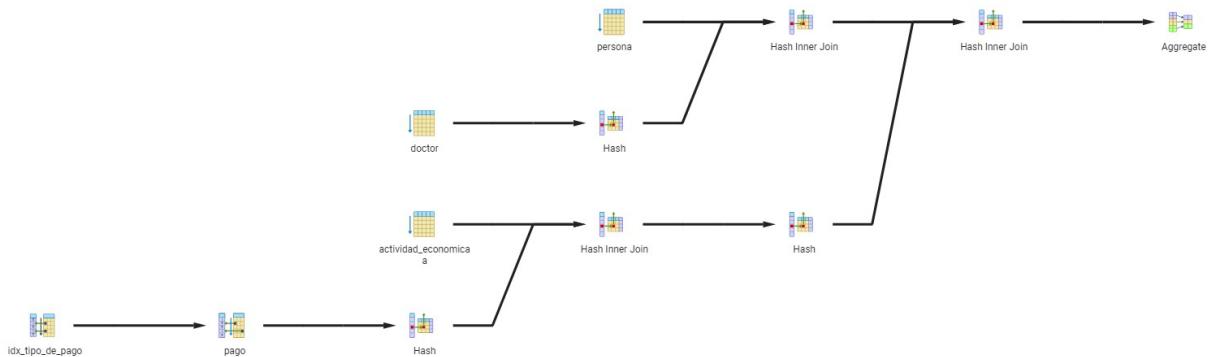


Figura 34: Query Plan Tree para la consulta 3 con índices en la tabla de 10 mil

QUERY PLAN	
text	
1	HashAggregate (cost=7769.09..7816.63 rows=4754 width=153) (actual time=256.066..257.915 rows=5020 loops=1)
2	Group Key: d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
3	Batches: 1 Memory Usage: 1745kB
4	-> Nested Loop (cost=0.86..7697.78 rows=4754 width=153) (actual time=2.073..249.553 rows=5020 loops=1)
5	-> Nested Loop (cost=0.58..6217.78 rows=4754 width=143) (actual time=1.613..184.837 rows=5020 loops=1)
6	-> Nested Loop (cost=0.29..4063.75 rows=4754 width=121) (actual time=0.883..69.620 rows=5020 loops=1)
7	-> Seq Scan on pago pg (cost=0.00..433.77 rows=4754 width=22) (actual time=0.298..6.633 rows=5020 loops=1)
8	Filter: ((tipo_de_pago)::text = 'Efectivo'::text)
9	Rows Removed by Filter: 14980
10	-> Index Scan using pk_actividaddeconomica_codigo on actividad_economica ae (cost=0.29..0.76 rows=1 width=119) (actual time=0.012..0.012 rows=1 loops=...)
11	Index Cond: ((codigo)::text = (pg.aecodigo)::text)
12	-> Index Scan using pk_persona_dni on persona p (cost=0.29..0.45 rows=1 width=22) (actual time=0.022..0.022 rows=1 loops=5020)
13	Index Cond: ((dni)::text = (ae.ddni)::text)
14	-> Index Scan using pk_doctor_pdni on doctor d (cost=0.29..0.31 rows=1 width=26) (actual time=0.012..0.012 rows=1 loops=5020)
15	Index Cond: ((pdni)::text = (p.dni)::text)
16	Planning Time: 13.808 ms
17	Execution Time: 258.325 ms

Figura 35: Query Plan para la consulta 3 sin índices en la tabla de 10 mil

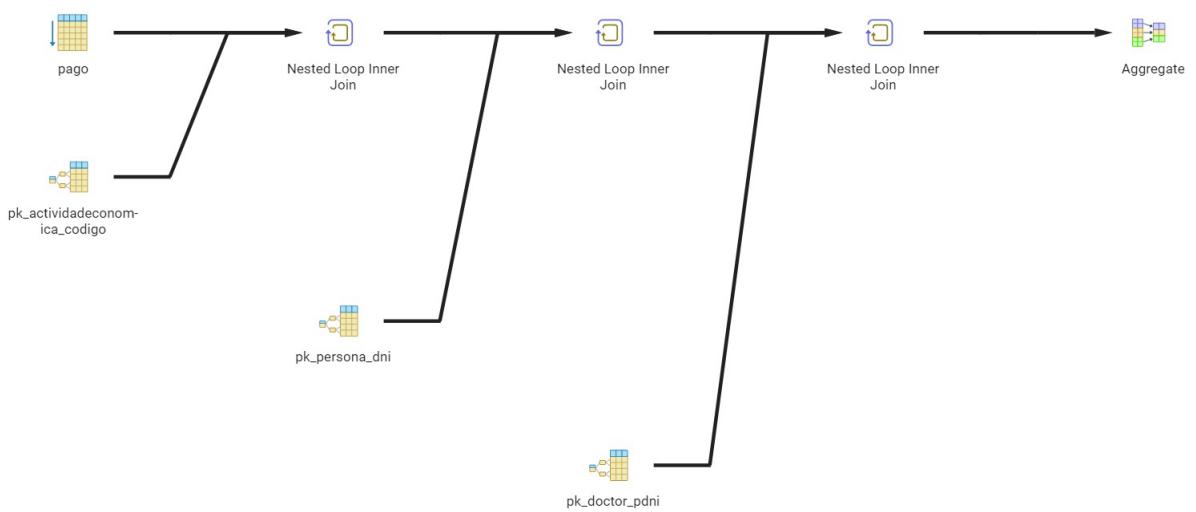


Figura 36: Query Plan Tree para la consulta 3 sin índices en la tabla de 10 mil

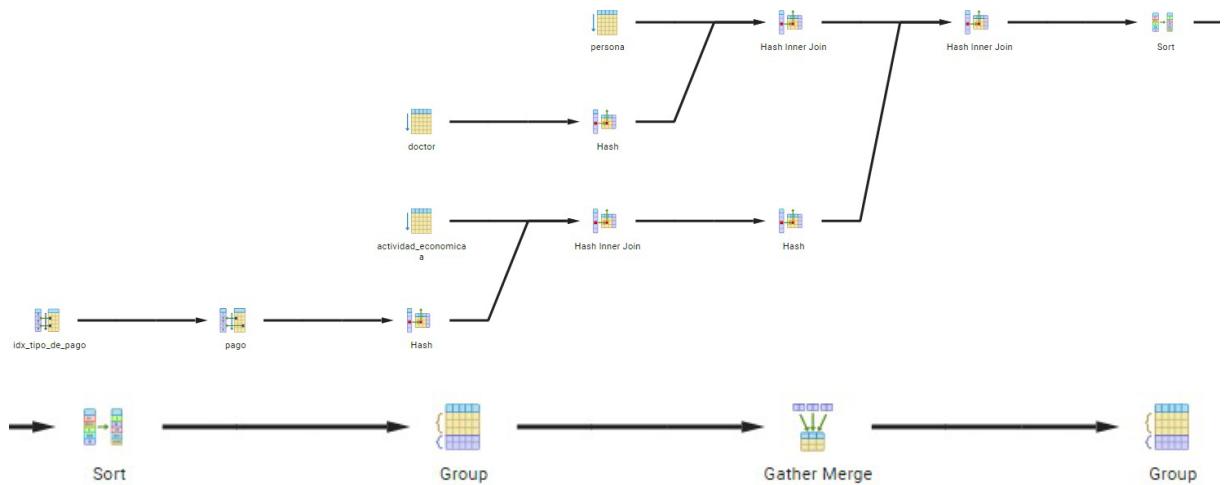


Figura 37: Query Plan Tree para la consulta 3 con índices en la tabla de 100 mil

QUERY PLAN	
text	
1	Group (cost=22819.27..28709.43 rows=50787 width=153) (actual time=530.265..620.528 rows=49774 loops=1)
2	Group Key: d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
3	-> Gather Merge (cost=22819.27..28074.60 rows=42322 width=153) (actual time=530.264..604.473 rows=49774 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Group (cost=21819.25..22189.57 rows=21161 width=153) (actual time=481.537..490.255 rows=16591 loops=3)
7	Group Key: d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
8	-> Sort (cost=21819.25..21872.15 rows=21161 width=153) (actual time=481.532..484.134 rows=16591 loops=3)
9	Sort Key: d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
10	Sort Method: quicksort Memory: 3676kB
11	Worker 0: Sort Method: quicksort Memory: 3571kB
12	Worker 1: Sort Method: quicksort Memory: 3157kB
13	-> Parallel Hash Join (cost=12611.72..20298.92 rows=21161 width=153) (actual time=107.493..380.436 rows=16591 loops=3)
14	Hash Cond: ((p.dni)::text = (ae.ddni)::text)
15	-> Hash Join (cost=3135.00..10651.25 rows=41667 width=48) (actual time=50.030..302.686 rows=33333 loops=3)
16	Hash Cond: ((p.dni)::text = (d.pdni)::text)
17	-> Parallel Seq Scan on persona p (cost=0.00..6860.00 rows=250000 width=22) (actual time=0.013..172.696 rows=200000 loops=3)
18	-> Hash (cost=1885.00..1885.00 rows=100000 width=26) (actual time=49.772..49.773 rows=100000 loops=3)
19	Buckets: 131072 Batches: 1 Memory Usage: 6828kB
20	-> Seq Scan on doctor d (cost=0.00..1885.00 rows=100000 width=26) (actual time=0.785..18.901 rows=100000 loops=3)
21	-> Parallel Hash (cost=9212.21..9212.21 rows=21161 width=121) (actual time=56.445..56.449 rows=16591 loops=3)
22	Buckets: 65536 Batches: 1 Memory Usage: 8384kB
23	-> Parallel Hash Join (cost=3289.89..9212.21 rows=21161 width=121) (actual time=10.928..48.507 rows=16591 loops=3)
24	Hash Cond: ((ae.codigo)::text = (pg.aecodigo)::text)
25	-> Parallel Seq Scan on actividad_economica ae (cost=0.00..5485.34 rows=83334 width=119) (actual time=0.013..10.011 rows=66667 ...)
26	-> Parallel Hash (cost=2916.45..2916.45 rows=29875 width=22) (actual time=10.787..10.788 rows=16591 loops=3)
27	Buckets: 65536 Batches: 1 Memory Usage: 3264kB
28	-> Parallel Bitmap Heap Scan on pago pg (cost=574.02..2916.45 rows=29875 width=22) (actual time=2.451..16.928 rows=49774 lo...)
29	Recheck Cond: ((tipo_de_pago)::text = 'Efectivo'::text)
30	Heap Blocks: exact=1969
31	-> Bitmap Index Scan on idx_tipo_de_pago (cost=0.00..561.32 rows=50787 width=0) (actual time=2.155..2.155 rows=49774 lo...)
32	Index Cond: ((tipo_de_pago)::text = 'Efectivo'::text)
33	Planning Time: 4.179 ms
34	Execution Time: 624.503 ms
Total rows: 34 of 34 Ouvr complete 00:00:00.705	

Figura 38: Query Plan para la consulta 3 con índices en la tabla de 100 mil

QUERY PLAN	
text	
1	Finalize HashAggregate (cost=70836.02..72685.94 rows=50787 width=153) (actual time=1769.510..1802.506 rows=49774 loops=1)
2	Group Key: d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
3	Planned Partitions: 4 Batches: 5 Memory Usage: 8241kB Disk Usage: 7048kB
4	-> Gather (cost=61135.99..64422.24 rows=29875 width=153) (actual time=1695.682..1713.445 rows=49774 loops=1)
5	Workers Planned: 1
6	Workers Launched: 1
7	-> Partial HashAggregate (cost=60135.99..60434.74 rows=29875 width=153) (actual time=1664.977..1675.672 rows=24887 loops=2)
8	Group Key: d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
9	Batches: 1 Memory Usage: 7441kB
10	Worker 0: Batches: 1 Memory Usage: 7953kB
11	-> Nested Loop (cost=1.26..59687.86 rows=29875 width=153) (actual time=6.783..1626.196 rows=24887 loops=2)
12	-> Nested Loop (cost=0.84..46440.89 rows=29875 width=143) (actual time=5.079..1207.774 rows=24887 loops=2)
13	-> Nested Loop (cost=0.42..29274.73 rows=29875 width=121) (actual time=2.556..559.471 rows=24887 loops=2)
14	-> Parallel Seq Scan on pago pg (cost=0.00..3439.59 rows=29875 width=22) (actual time=0.684..93.869 rows=24887 loops=2)
15	Filter: ((tipo_de_pago)::text = 'Efectivo'::text)
16	Rows Removed by Filter: 75114
17	-> Index Scan using pk_actividaddeconomica_codigo on actividad_economica ae (cost=0.42..0.86 rows=1 width=119) (actual time=0.018..0.018 rows=1 loops=4...)
18	Index Cond: ((codigo)::text = (pg.aecodigo)::text)
19	-> Index Scan using pk_persona_dni on persona p (cost=0.42..0.57 rows=1 width=22) (actual time=0.025..0.025 rows=1 loops=49774)
20	Index Cond: ((dni)::text = (ae.ddni)::text)
21	-> Index Scan using pk_doctor_pdni on doctor d (cost=0.42..0.44 rows=1 width=26) (actual time=0.016..0.016 rows=1 loops=49774)
22	Index Cond: ((pdni)::text = (p.dni)::text)
23	Planning Time: 1.060 ms
24	Execution Time: 1809.508 ms

Figura 39: Query Plan para la consulta 3 sin índices en la tabla de 100 mil

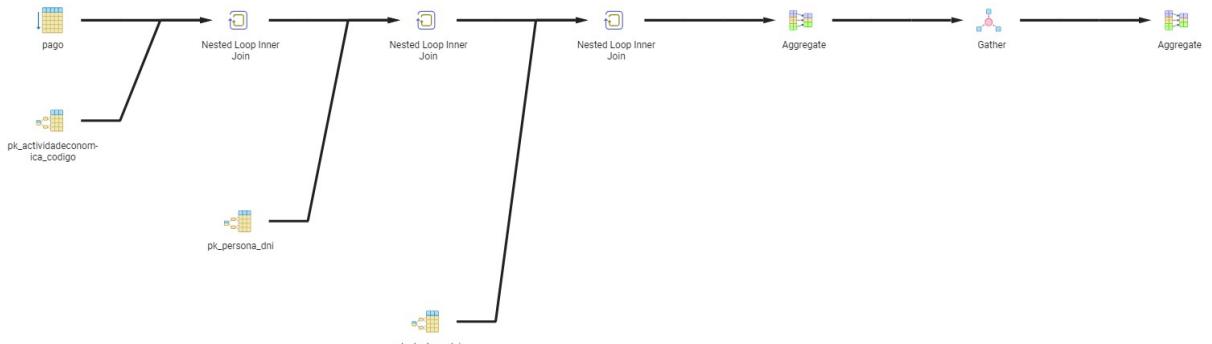


Figura 40: Query Plan Tree para la consulta 3 sin índices en la tabla de 100 mil

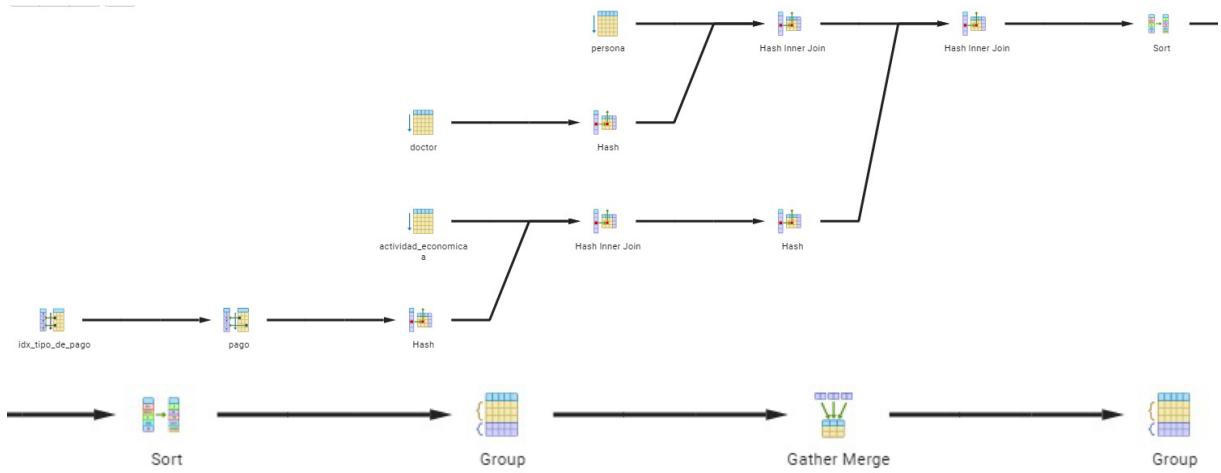


Figura 41: Query Plan Tree para la consulta 3 con índices en la tabla de 1 millón

QUERY PLAN	
	text
1	Group (cost=304642.94..362671.05 rows=500333 width=153) (actual time=5676.610..6874.842 rows=500184 loops=1)
2	Group Key: d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
3	-> Gather Merge (cost=304642.94..356416.89 rows=416944 width=153) (actual time=5676.606..6750.491 rows=500184 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Group (cost=303642.92..307291.18 rows=208472 width=153) (actual time=5560.812..5877.453 rows=166728 loops=3)
7	Group Key: d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
8	-> Sort (cost=303642.92..304164.10 rows=208472 width=153) (actual time=5560.807..5829.244 rows=166728 loops=3)
9	Sort Key: d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
10	Sort Method: external merge Disk: 27752kB
11	Worker 0: Sort Method: external merge Disk: 26160kB
12	Worker 1: Sort Method: external merge Disk: 28704kB
13	-> Parallel Hash Join (cost=148626.68..268834.45 rows=208472 width=153) (actual time=4078.839..4361.428 rows=166728 loops=3)
14	Hash Cond: ((p.dni)::text = (ae.ddni)::text)
15	-> Parallel Hash Join (cost=21075.00..128380.50 rows=416667 width=48) (actual time=1423.569..2158.559 rows=333333 loops=3)
16	Hash Cond: ((p.dni)::text = (d.pdni)::text)
17	-> Parallel Seq Scan on persona p (cost=0.00..68596.00 rows=2500000 width=22) (actual time=0.838..473.748 rows=2000000 loops=3)
18	-> Parallel Hash (cost=13017.67..13017.67 rows=416667 width=26) (actual time=218.413..218.414 rows=333333 loops=3)
19	Buckets: 131072 Batches: 16 Memory Usage: 4896kB
20	-> Parallel Seq Scan on doctor d (cost=0.00..13017.67 rows=416667 width=26) (actual time=0.892..85.090 rows=333333 loops=3)
21	-> Parallel Hash (cost=121076.77..121076.77 rows=208472 width=121) (actual time=1745.354..1745.358 rows=166728 loops=3)
22	Buckets: 65536 Batches: 16 Memory Usage: 5504kB
23	-> Parallel Hash Join (cost=31717.81..121076.77 rows=208472 width=121) (actual time=1226.354..1620.190 rows=166728 loops=3)
24	Hash Cond: ((ae.codigo)::text = (pg.aecodigo)::text)
25	-> Parallel Seq Scan on actividad_economica ae (cost=0.00..54845.33 rows=833333 width=119) (actual time=0.680..356.920 rows=666667 l...)
26	-> Parallel Hash (cost=27889.91..27889.91 rows=208472 width=22) (actual time=210.946..210.947 rows=166728 loops=3)
27	Buckets: 131072 Batches: 4 Memory Usage: 7936kB
28	-> Parallel Bitmap Heap Scan on pago pg (cost=5606.01..27889.91 rows=208472 width=22) (actual time=9.230..145.966 rows=166728 lo...
29	Recheck Cond: ((tipo_de_pago)::text = 'Efectivo'::text)
30	Heap Blocks: exact=9611
31	-> Bitmap Index Scan on idx_tipo_de_pago (cost=0.00..5480.92 rows=500333 width=0) (actual time=23.725..23.725 rows=500184 loo...
32	Index Cond: ((tipo_de_pago)::text = 'Efectivo'::text)
33	Planning Time: 4.012 ms
34	Execution Time: 6900.799 ms
Total rows: 34 of 34 Query complete 00:00:06.003	

Figura 42: Query Plan para la consulta 3 con índices en la tabla de 1 millón

QUERY PLAN	
1	HashAggregate (cost=587009.78..614489.01 rows=500333 width=153) (actual time=15675.290..16022.524 rows=500184 loops=1)
2	Group Key: d.pdni, p.nombre, p.apellido, ae.descripcion, pg.monto, pg.fecha_pago
3	Planned Partitions: 32 Batches: 33 Memory Usage: 8209kB Disk Usage: 95832kB
4	-> Gather (cost=1001.28..479594.54 rows=500333 width=153) (actual time=2.221..14894.063 rows=500184 loops=1)
5	Workers Planned: 2
6	Workers Launched: 2
7	-> Nested Loop (cost=1.28..428561.24 rows=208472 width=153) (actual time=4.606..15282.098 rows=166728 loops=3)
8	-> Nested Loop (cost=0.86..334562.46 rows=208472 width=143) (actual time=3.291..10802.707 rows=166728 loops=3)
9	-> Nested Loop (cost=0.43..213221.34 rows=208472 width=121) (actual time=1.864..4312.911 rows=166728 loops=3)
10	-> Parallel Seq Scan on pago pg (cost=0.00..30094.67 rows=208472 width=22) (actual time=0.700..300.633 rows=166728 loops=3)
11	Filter: ((tipo_de_pago)::text = 'Efectivo'::text)
12	Rows Removed by Filter: 499939
13	-> Index Scan using pk_actividadeconomico_codigo on actividad_economica ae (cost=0.43..0.88 rows=1 width=119) (actual time=0.023..0.023 rows=1 loops=50...)
14	Index Cond: ((codigo)::text = (pg.aecodigo)::text)
15	-> Index Scan using pk_persona_dni on persona p (cost=0.43..0.58 rows=1 width=22) (actual time=0.038..0.038 rows=1 loops=500184)
16	Index Cond: ((dni)::text = (ae.ddni)::text)
17	-> Index Scan using pk_doctor_pdni on doctor d (cost=0.42..0.45 rows=1 width=26) (actual time=0.026..0.026 rows=1 loops=500184)
18	Index Cond: ((pdni)::text = (p.dni)::text)
19	Planning Time: 0.624 ms
20	Execution Time: 16048.440 ms

Figura 43: Query Plan para la consulta 3 sin índices en la tabla de 1 millón

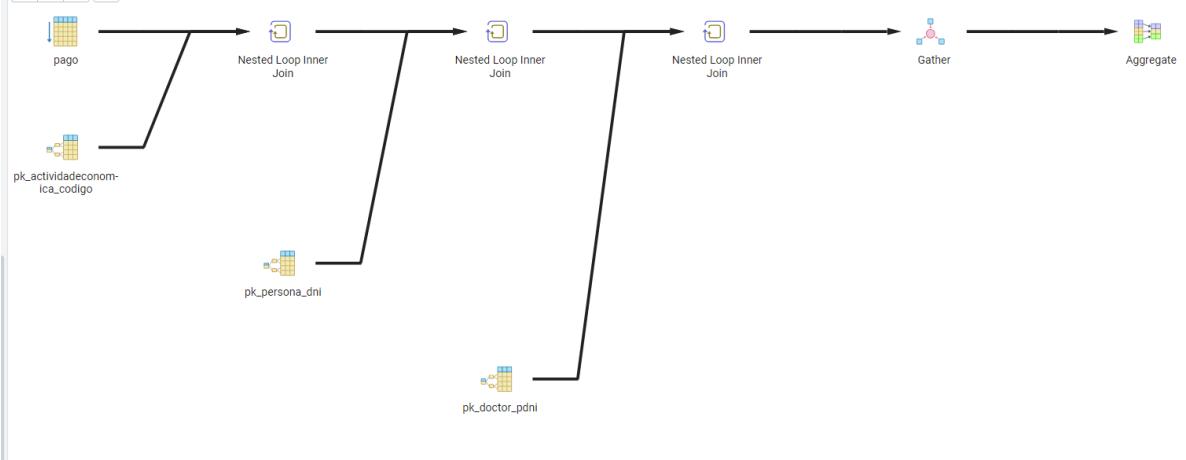


Figura 44: Query Plan para la consulta 3 sin índices en la tabla de 1 millón

5.5.4. Sin índices

Consulta 1				
Ejecución	1k	10k	100k	1m
1	15.598	143.699	1212.732	20571.106
2	8.615	121.285	1121.699	23857.599
3	8.859	124.977	1141.455	20566.474
4	9.453	118.016	1125.272	20696.327
5	8.452	147.543	1167.438	20303.165
Promedio	10.2	131.104	1152.91	21498.93

Consulta 2				
Ejecución	1k	10k	100k	1m
1	12.142	292.69	1122.88	34765.75
2	12.658	164.819	1176.59	31866.58
3	12.245	170.038	1240.18	39514.342
4	12.568	177.168	1144.168	39289.871
5	11.196	170.567	1149.56	31295.75
Promedio	12.14	195.1	1166.67	35345.8

Consulta 3			
Ejecución	10k	100k	1m
1	258.325	1809.587	15846.65
2	142.579	1804.033	16231.45
3	143.563	1805.457	15489.321
4	150.478	1845.781	15568.146
5	164.587	1876.772	15489.398
Promedio	171.906	1820.3	15724.99

5.5.5. Con índices

Consulta 1				
Ejecución	1k	10k	100k	1m
1	2.401	17.136	294.895	3879.26
2	2.626	17.006	261.055	3711.503
3	2.328	17.022	254.682	3720.47
4	2.447	17.071	288.876	3920.214
5	2.453	17.580	290.694	3800.156
Promedio	2.42	17.102	277.402	3806.12

Consulta 2				
Ejecución	1k	10k	100k	1m
1	2.750	28.466	440.765	9290.04
2	2.737	25.517	379.19	5612.499
3	2.779	25.762	403.038	9013.227
4	2.658	25.781	370.168	9335.573
5	2.986	25.107	393.567	9159.625
Promedio	2.782	26.12	397.34	9082.19

Consulta 3			
Ejecución	10k	100k	1m
1	54.933	640.122	6900.799
2	40.758	529.247	7000.654
3	40.706	607.954	6874.589
4	45.806	555.892	7152.297
5	43.764	580.078	6982.459
Promedio	45.193	582.65	6982.16

5.6. Resultados

5.6.1. Consulta 1



Figura 45: Comparación de la consultan 1

5.6.2. Consulta 2

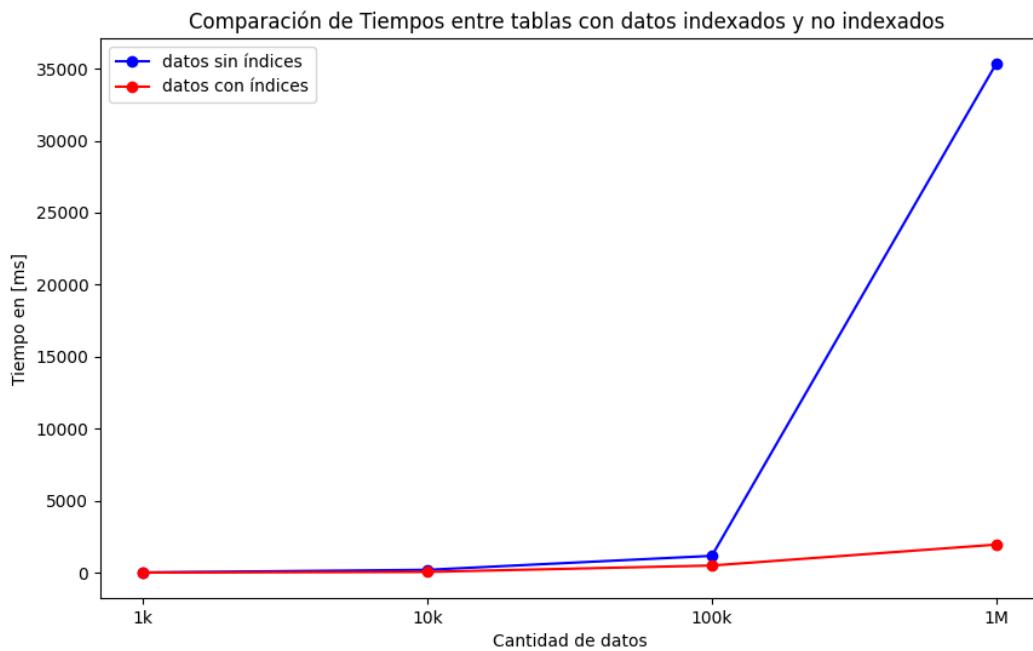


Figura 46: Comparación de la consulta 2

5.6.3. Consulta 3

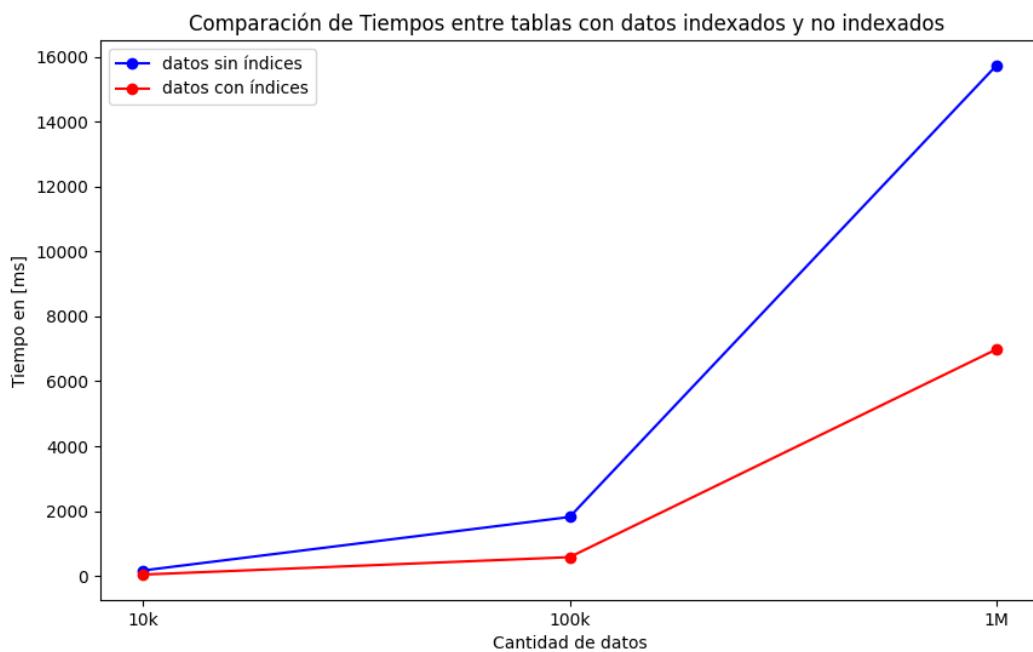


Figura 47: Comparación de la consulta 3

5.7. Análisis y Discusión

Las tres gráficas son muy similares. La duración en los tres ejemplos comienza pegado, sin embargo apenas va aumentando el número de datos, se separan, habiendo una clara separación a partir de los cien mil datos. Algo a tomar en cuenta es que se utilizó índices btree en las tres consultas, ya que en ningún caso, el índice hash era identificado. Cuando se utilizaba el índice hash, PostgreSQL hacía búsquedas secuenciales, en vez de utilizar el índice. Sin embargo, a la hora de usar un índice btree, PostgreSQL sí utilizaba el índice btree, dejando de lado la búsqueda secuencial.

Un dato adicional que se pudo comprobar durante la experimentación con los índices es la disparidad en el tiempo a la hora de crear los índices btree y hash, siendo el tiempo de creación del último mucho mayor. Esto seguramente al hecho de tener que computar el valor hash para cada valor en la columna.

5.7.1. Comparación índices vs no índices

En las querys sin índices, al haber desactivado la generación automática de índices, el explain analyze muestra que se usan entre tres a cinco nested loops (dependiendo de la query) en los que se identifican index scans sobre las llaves primarias.

Por otro lado, cuando se va a hacer las querys con índices y se activa la creación de índices de parte de PostgreSQL, se puede identificar que pgadmin utiliza hash-joins y también índices hash a la hora de hacer las querys. Esto debido a que al ser joins entre llaves primarias, habrá un montón de tuplas tanto en R y S que satisfacen el join. Por ejemplo, en el join que se hace en la query 3 entre persona y doctor, S, que sería la tabla con menos tuplas, sería doctor, que tiene un millón de tuplas y como doctor es hijo de persona, todo el millón de tuplas va a satisfacer el join con persona. Por lo tanto es beneficioso utilizar el hash join, como lo hace pg admin.

Concluimos nuestra discusión sobre la optimización del proyecto mencionando la búsqueda que se utiliza en los índices que fueron agregados adicionalmente en cada consulta.

Los índices no son una varita mágica que se pueden utilizar en todas las ocasiones. El índice es más útil cuando se buscará una pequeña porción de los datos, sin embargo si se va a buscar y comparar una gran porción de los datos, PostgreSQL prefiere realizar una búsqueda secuencial. Sin embargo, ¿Qué pasa si se busca suficientes datos en donde un index scan no vale la pena, pero son muy pocos datos para que una búsqueda secuencial sea eficiente? Se utiliza el tipo de búsqueda que se utilizan en nuestras querys, un bitmap index scan. El beneficio del bitmap scan es que itera sobre todos los índices, generando un bitmap con unos y ceros, significando que ahí se encuentra información que se quiere. Una vez terminado ello, se hace el heap scan para conseguir la información que se requiere.

6. Conclusiones

- Primero plantear las reglas semánticas, crear un modelo Entidad-Relacion y transformar al Modelo Relacional nos permitió crear las tablas de la base de datos con

mayor facilidad.

- Las condiciones como por ejemplo 1 a N (dos lineas en el Modelo ER) se pierden a la hora de crear las tablas de SQL.
- La creación de stored procedures para el manejo del stock es importante para mantener un correcto uso de la base de datos. Estos stored procedures permiten que la base de datos sea escalable a largo plazo.
- Se pudo diferenciar los diferentes tipos de índices y búsquedas, comparando y concluyendo estrategias de planificación de índices acorde a lo buscado en las consultas.

7. Anexo

7.1. Link al Drawio del Modelo de Entidad Relación

[Link](#)

7.2. Github

Adicional al informe, se creó una repo de github. [Repo de Github](#)

7.3. Data_{Dump}

[Data dump](#)

7.4. Script para poblar base de datos

```
1 import pandas as pd
2 from faker import Faker
3 import random
4
5 fake = Faker()
6
7 personas = []
8 unique_documents = set()
9
10 # Creacion de personas -----
11
12 for i in range(6000):
13     while True:
14         nrodocumento = fake.unique.random_number(digits=8)
15         if nrodocumento not in unique_documents:
16             unique_documents.add(nrodocumento)
```

```

17         break
18     nombre = fake.first_name()
19     apellido = fake.last_name()
20     fecha_de_nacimiento = fake.date_of_birth(minimum_age=30, maximum_age=60)
21     personas.append({
22         'dni': nrodocumento,
23         'nombre': nombre,
24         'apellido': apellido,
25         'fecha_de_nacimiento': fecha_de_nacimiento
26     })
27
28 df = pd.DataFrame(personas)
29 df.to_csv('personas_mil.csv', index=False)
30 print('personas creadas')
31
32 # Creacion de doctor -----
33
34 doctores = []
35 especialidades = [
36     'Cardiología', 'Dermatología', 'Endocrinología',
37     'Gastroenterología', 'Geriatría', 'Hematología',
38     'Inmunología', 'Infectología', 'Medicina Interna',
39     'Nefrología', 'Neumología', 'Neurología',
40     'Obstetricia y Ginecología', 'Oncología', 'Oftalmología',
41     'Otorrinolaringología', 'Pediatría', 'Psiquiatría',
42     'Radiología', 'Reumatología', 'Urología',
43     'Cirugía General', 'Cirugía Cardiovascular', 'Cirugía Plástica',
44     'Cirugía Torácica', 'Cirugía Vascular', 'Medicina de Emergencias',
45     'Medicina Física y Rehabilitación', 'Medicina del Deporte', 'Medicina del Trabajo'
46 ]
47
48
49
50 for i in range(1000):
51
52     sueldo = round(random.uniform(30000, 300000), 2)
53     especialidad = random.choice(especialidades)
54
55     doctores.append({
56         'pdni': personas[i]['dni'],
57         'sueldo': sueldo,

```

```

58         'especialidad': especialidad
59     })
60
61 df = pd.DataFrame(doctores)
62 df.to_csv('doctor_mil.csv', index=False)
63 print('doctor exitoso')
64
65 # Creacion de interno -----
66
67 internos = []
68
69 for i in range(1000,2000):
70     sueldo = round(random.uniform(1025, 3000), 2)
71
72     internos.append({
73         'pdni': personas[i]['dni'],
74         'sueldo': sueldo
75     })
76
77 df = pd.DataFrame(internos)
78 df.to_csv('internos_mil.csv', index=False)
79 print('internos exitoso')
80
81 # Creacion de enfermera -----
82
83 enfermeras = []
84
85 for i in range(2000, 3000):
86     sueldo = round(random.uniform(3000, 6000), 2)
87
88     enfermeras.append({
89         'pdni': personas[i]['dni'],
90         'sueldo': sueldo
91     })
92
93 df = pd.DataFrame(enfermeras)
94 df.to_csv('enfermeras_mil.csv', index=False)
95 print('enfermeras exitoso')
96
97 # Creacion de secretaria -----
98

```

```

99 secretarias = []
100
101 for i in range(3000, 4000):
102
103     sueldo = round(random.uniform(2000, 5000), 2)
104
105     secretarias.append({
106         'pdni': personas[i]['dni'],
107         'sueldo': sueldo
108     })
109
110 df = pd.DataFrame(secretarias)
111 df.to_csv('secretarias_mil.csv', index=False)
112 print('secretarias exitoso')
113
114 # Creacion de gerente -----
115
116 gerentes = []
117
118 for i in range(4000, 5000):
119
120     sueldo = round(random.uniform(15000, 30000), 2)
121
122     gerentes.append({
123         'pdni': personas[i]['dni'],
124         'sueldo': sueldo,
125     })
126
127 df = pd.DataFrame(gerentes)
128 df.to_csv('gerentes_mil.csv', index=False)
129 print('gerentes exitoso')
130
131
132 # Creacion de paciente -----
133
134 pacientes = []
135
136 for i in range(5000, 6000):
137     nrodocumento = personas[i]['dni']
138
139     pacientes.append({

```

```

140         'pdni': nrodocumento
141     })
142
143 df = pd.DataFrame(pacientes)
144 df.to_csv('pacientes_mil.csv', index=False)
145 print('pacientes exitoso')
146
147
148 # Creacion de insumos medicos
149 insumos_medicos = []
150 insumos_codigos = set()
151
152 for i in range(1000):
153     while True:
154         codigo = fake.unique.random_number(digits=10)
155         if codigo not in insumos_codigos:
156             insumos_codigos.add(codigo)
157             break
158         material = fake.word()
159         marca = fake.company()
160         costo_por_unidad = round(random.uniform(10, 100), 2)
161         nombre_insumo = fake.word()
162         cantidad = random.randint(1, 100)
163
164         insumos_medicos.append({
165             'nro_serial': codigo,
166             'material': material,
167             'marca': marca,
168             'costo_por_unidad': costo_por_unidad,
169             'nombre': nombre_insumo,
170             'cantidad': cantidad
171         })
172
173 df = pd.DataFrame(insumos_medicos)
174 df.to_csv('insumos_medicos_mil.csv', index=False)
175 print('insumos medicos exitoso')
176
177 # Creacion de Pedido
178 pedidos = []
179
180 for i in range(1000):

```

```

181     while True:
182         codigo = fake.unique.random_number(digits=10)
183         if codigo not in insumos_codigos: # ensure unique order codes
184             insumos_codigos.add(codigo)
185             break
186         fecha_pedido = fake.date_this_year()
187         nrodocumento = random.choice(gerentes)['pdni']
188
189         pedidos.append({
190             'codigo': codigo,
191             'gdni': nrodocumento,
192             'fecha_pedido': fecha_pedido
193         })
194
195     df = pd.DataFrame(pedidos)
196     df.to_csv('pedidos_mil.csv', index=False)
197     print('pedido exitoso')
198
199 # Creacion de detallePedido
200 detallePedido = []
201 insumos_utilizados = set()
202
203 def escogerPedido():
204     return random.choice(pedidos)['codigo']
205
206 def escogerInsumoMedico():
207     while True:
208         insumoMedico = random.choice(insumos_medicos)
209         if insumoMedico['nro_serial'] not in insumos_utilizados:
210             insumos_utilizados.add(insumoMedico['nro_serial'])
211             return insumoMedico['nro_serial']
212
213 for i in range(1000):
214     pcodigo = escogerPedido()
215     insumoMedico = escogerInsumoMedico()
216     cantidad = random.randint(1, 100)
217
218     detallePedido.append({
219         'im_ns': insumoMedico,
220         'pcodigo': pcodigo,
221         'cantidad': cantidad

```

```

222     })
223
224     df = pd.DataFrame(detallePedido)
225     df.to_csv('detallePedido_mil.csv', index=False)
226     print('detallePedido exitoso')
227
228 # Creacion de actividad_economica -----
229
230 actividad_economica = []
231 used_codigos = set()
232 used_fecha_hora = {}
233
234 for i in range(2000):
235     while True:
236         codigo = fake.unique.random_number(digits=10)
237         if codigo not in used_codigos:
238             used_codigos.add(codigo)
239             break
240
241         pnrodoc = random.choice(pacientes)['pdni']
242         dnrodoc = random.choice(doctores)['pdni']
243         inrodoc = random.choice(internos)['pdni']
244         descripcion = ''
245
246         while True:
247             fecha_ae = fake.date_this_year()
248             horaInicio = fake.time()
249             if fecha_ae not in used_fecha_hora:
250                 used_fecha_hora[fecha_ae] = {horaInicio}
251                 break
252             elif horaInicio not in used_fecha_hora[fecha_ae]:
253                 used_fecha_hora[fecha_ae].add(horaInicio)
254                 break
255         monto = round(random.uniform(10000, 40000), 2)
256
257         actividad_economica.append({
258             'pdni': pnrodoc,
259             'ddni': dnrodoc,
260             'idni': inrodoc,
261             'codigo': codigo,

```

```

263     'descripcion': descripcion,
264     'fecha_ae': fecha_ae,
265     'horainicio': horaInicio,
266     'monto': monto
267 )
268
269 df = pd.DataFrame(actividad_economica)
270 df.to_csv('actividad_economica_mil.csv', index=False)
271 print('actividad_economica exitoso')
272
273 # Creacion de operacion -----
274
275 operaciones = []
276
277 for i in range(1000):
278     aecodigo = actividad_economica[i]['codigo']
279     enrodoc = random.choice(enfermeras)['pdni']
280     tipo_operacion = fake.word()
281
282     operaciones.append({
283         'aecodigo': aecodigo,
284         "edni": enrodoc,
285         "tipo_operacion": tipo_operacion
286     })
287
288
289 df = pd.DataFrame(operaciones)
290 df.to_csv('operaciones_mil.csv', index=False)
291 print('operacion exitoso')
292
293 # Creacion de consulta
294
295 consulta = []
296
297 for i in range(1000,2000):
298     aecodigo = actividad_economica[i]['codigo']
299     n = i-1000
300     snrodoc = secretarias[n]['pdni']
301     tipo_operacion = fake.word()
302
303     consulta.append({

```

```

304     'aecodigo':aecodigo,
305     "sdni":snrodoc,
306     "duracion":30
307 )
308
309
310 df = pd.DataFrame(consulta)
311 df.to_csv('consulta_mil.csv', index=False)
312 print('consulta exitoso')
313
314 # Creacion de utiliza
315
316 utiliza = []
317 used_combinations = set()
318
319 for i in range(5000):
320
321     while True:
322         enrodoc = random.choice(enfermeras)['pdni']
323         ocodigo = random.choice(operaciones)['aecodigo']
324         im_ns = random.choice(insumos_medicos)['nro_serial']
325         combination = (ocodigo, im_ns, enrodoc)
326
327         if combination not in used_combinations:
328             used_combinations.add(combination)
329             break
330
331         cantidad = random.randint(1, 15)
332
333         utiliza.append({
334             'ocodigo': ocodigo,
335             'im_ns': im_ns,
336             'cantidad': cantidad,
337             'edni': enrodoc,
338         })
339
340 df = pd.DataFrame(utiliza)
341 df.to_csv('utiliza_mil.csv', index=False)
342 print('utiliza exitoso')
343
344 # Creacion de pago

```

```

345
346 pago = []
347 used_codigos = set()
348
349 for i in range(2000):
350     aecodigo = actividad_economica[i]['codigo']
351     while True:
352         codigo = fake.unique.random_number(digits=10)
353         if codigo not in used_codigos:
354             used_codigos.add(codigo)
355             break
356     monto = actividad_economica[i]['monto']
357     tipo_de_pago = fake.random_element(elements=('Tarjeta de Credito', 'Tarjeta de Debito',
358                                         'Seguro', 'Efectivo'))
359     fecha_pago = fake.date_this_year()
360
361     pago.append({
362         'codigo': codigo,
363         'aecodigo': aecodigo,
364         'monto': monto,
365         'tipo_de_pago': tipo_de_pago,
366         'fecha_de_pago': fecha_pago
367     })
368
369 df = pd.DataFrame(pago)
370 df.to_csv('pago_mil.csv', index=False)
371 print('pago exitoso')
372

```