A large, light gray Fibonacci spiral is centered on the page, starting from a small square in the upper right and expanding outwards in a clockwise direction. It passes through the text and author information.

Informe Técnico

INVENTARIO GLOBAL - BASE DE DATOS DISTRIBUIDA AVANZADA CON SQL SERVER

Anthony Heriberto Montero Roman
2019275097

Kun Kin Zheng Liang
2022205015

ÍNDICE

1. Introducción	4
2. Diseño de la Solución	4
2.1. Esquema de la base de datos	4
2.2. Estrategia de fragmentación horizontal y vertical	6
2.2.1. Fragmentación Horizontal	6
2.2.2. Fragmentación Vertical	7
2.3. Plan de replicación	8
2.3.1. Replicación Transaccional	8
2.3.2. Replicación Combinacional	8
2.4. Estrategia de respaldo y recuperación	9
3. Configuraciones Técnicas	9
3.1. Configuración de las instancias de SQL Server	9
3.2. Implementación de la fragmentación	10
3.2.1. Fragmentación horizontal	10
3.2.2. Fragmentación vertical	10
3.3. Configuración de la replicación	11
3.4. Configuración de backups automáticos	12
3.5. Configuración de niveles de aislamiento	13
4. Pruebas y Resultados	17
4.1. Pruebas de fragmentación	17
4.1.1. Fragmentación Horizontal	17
4.1.2. Fragmentación Vertical	17
4.2. Pruebas de recuperación ante fallos	18
4.3. Resultados de consultas distribuidas	19
5. Pruebas de rendimiento	21
6. Conclusiones y Recomendaciones	22
6.1. Conclusiones	22

6.2. Recomendaciones	23
--------------------------------	----

INTRODUCCIÓN

En un mundo globalizado donde las empresas gestionan volúmenes crecientes de datos distribuidos geográficamente, el diseño e implementación de bases de datos distribuidas se ha convertido en una necesidad estratégica para garantizar alta disponibilidad, escalabilidad y resiliencia. Este proyecto tiene como objetivo abordar estos desafíos a través del desarrollo de una base de datos distribuida avanzada utilizando Microsoft SQL Server, integrando técnicas y conceptos fundamentales como fragmentación, replicación, respaldo, concurrencia y consistencia.

El contexto práctico del proyecto se centra en las necesidades de una empresa global de gestión de inventarios. La solución diseñada debe ser capaz de operar de manera eficiente bajo condiciones diversas, asegurando la integridad de los datos y su disponibilidad en tiempo real, incluso ante fallos en los sistemas o incrementos significativos en la demanda de acceso. La implementación requerirá un enfoque general que combine estrategias de diseño lógico, configuración avanzada de SQL Server y simulaciones para validar el desempeño del sistema.

El desarrollo de esta base de datos distribuida no solo permitirá aplicar conceptos teóricos en un entorno realista, sino que también proporcionará una experiencia práctica valiosa al enfrentar problemas complejos como la sincronización de datos entre regiones, el manejo de transacciones concurrentes y la recuperación ante desastres. A través de este proyecto, se evaluarán las capacidades técnicas para implementar soluciones robustas y escalables que respondan a los estándares modernos de disponibilidad y eficiencia.

DISEÑO DE LA SOLUCIÓN

2.1 ESQUEMA DE LA BASE DE DATOS

En la Figura 1 se presenta el modelo entidad-relación (MER). Este modelo se estructura en torno a cinco entidades principales que interactúan entre sí de manera coherente y organizada.

La entidad Región constituye la base geográfica del sistema, almacenando las ubicaciones donde se gestiona el inventario. Cada región se identifica mediante un `regionID` único y contiene información esencial como el nombre, país y zona horaria correspondiente. A través de la relación “tiene”, cada región puede estar vinculada con múltiples usuarios y almacenes, estableciendo así una estructura organizativa clara.

En cuanto a la entidad Usuario, esta representa a los responsables que interactúan con el sistema para realizar pedidos. Cada usuario se identifica por un `usuarioID` único y mantiene datos personales como nombre, apellido, email y teléfono. La relación “realiza” establece el vínculo entre los usuarios y sus pedidos correspon-

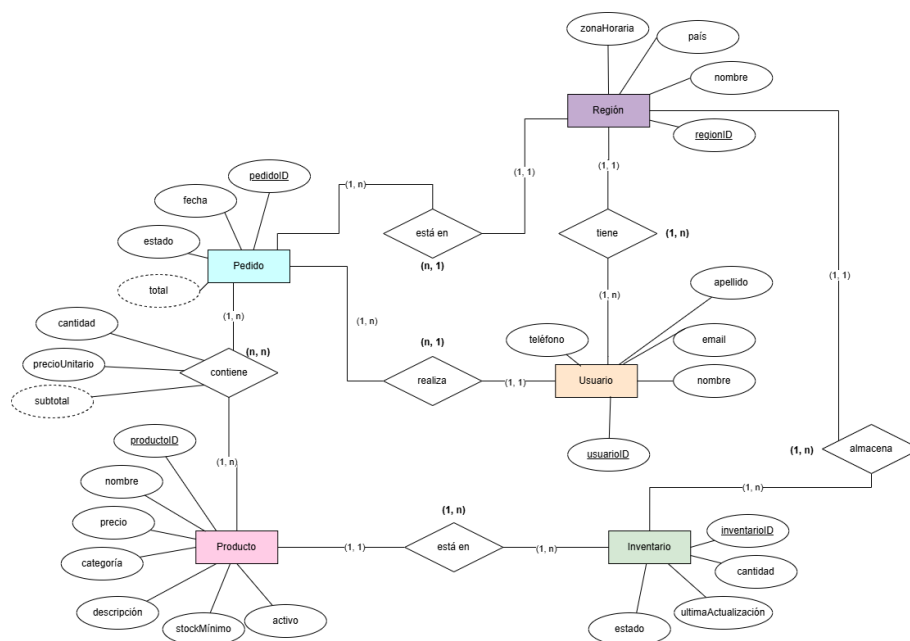


Figura 1: Modelo Entidad Relación (MER).

dientes.

Los Pedidos se gestionan como una entidad separada que registra las transacciones del sistema. Cada pedido se identifica mediante un *pedidoID* y almacena información crucial como la fecha, el estado y el total de la transacción. La relación “contiene” permite detallar qué productos específicos forman parte de cada pedido.

La entidad Producto abarca todos los bienes disponibles en el sistema. Cada producto se identifica por un *productoID* y cuenta con atributos detallados como nombre, precio, categoría, descripción, stock mínimo y un indicador de si está activo. Mediante la relación “está en”, los productos se conectan con el inventario específico de cada región.

El Inventario, como entidad final, mantiene el control del estado actual de los productos almacenados. Se identifica por un *inventarioID* y registra la cantidad disponible, la última actualización y el estado actual de cada producto en las diferentes ubicaciones.

En la Figura 2 se encuentra el modelo físico implementado en SQL Server que representa la materialización concreta del MER en estructuras de base de datos; este modelo se compone de seis tablas principales diseñada para mantener la integridad de los datos.

La tabla Region constituye una base fundamental del sistema, incorporando los campos *regionID*, nombre, país y *zonaHoraria*. Su diseño facilita una fragmentación horizontal basada en regiones geográficas, lo que permite una localización eficiente de los datos según su distribución territorial.

En cuanto a la tabla Usuario, esta almacena información personal y de contacto a través de los campos *usuarioID*, nombre, apellido, email, teléfono y *regionID*. La integridad referencial se mantiene mediante una

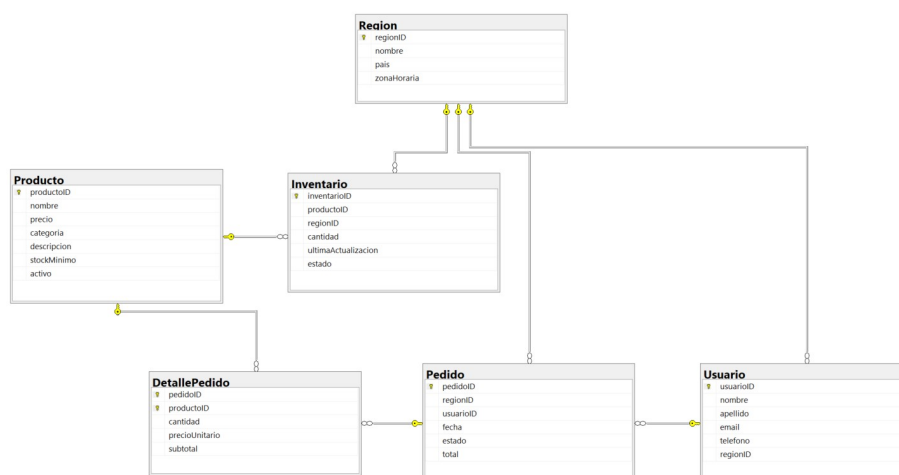


Figura 2: Modelo Físico.

clave foránea que establece una conexión con la tabla Region, asegurando que cada usuario esté correctamente asociado a una región específica.

La tabla Pedido se estructura con los campos pedidoID, fecha, estado, total, usuarioID y regionID. Una característica notable de esta tabla es su implementación de replicación en diferentes nodos, lo cual garantiza la consistencia de las órdenes entre las distintas regiones del sistema.

DetallePedido materializa la relación “contiene” del MER, incluyendo los campos pedidoID, productoID, cantidad, precioUnitario y subtotal. Esta tabla ha sido diseñada siguiendo principios de normalización para evitar redundancias y mantener la integridad de los datos de cada pedido.

La tabla Producto mantiene un registro completo de los artículos disponibles, con campos como productoID, nombre, precio, categoría, descripción, stockMínimo y activo. Su estructura ha sido optimizada mediante índices específicos para mejorar el rendimiento de las consultas frecuentes.

Finalmente, la tabla Inventario gestiona el control de productos por región, incluyendo campos como inventarioID, productoID, regionID, cantidad, ultimaActualización y estado.

2.2 ESTRATEGIA DE FRAGMENTACIÓN HORIZONTAL Y VERTICAL

2.2.1. Fragmentación Horizontal

La fragmentación horizontal implementada en este proyecto se diseñó para dividir los datos de la tabla Región en dos instancias de SQL Server basadas en rangos geográficos. Esta estrategia permite distribuir los datos de manera eficiente, mejorar la escalabilidad del sistema y garantizar un acceso optimizado a los datos según la ubicación.

La división de datos en el sistema sigue un esquema de fragmentación geográfica claramente definido para

la tabla Región, implementado a través de dos servidores principales con reglas específicas de distribución.

El Servidor 1, dedicado a América, está configurado para manejar exclusivamente las regiones correspondientes a países del continente americano. Este servidor tiene implementada una restricción específica que permite el registro únicamente de países predefinidos, incluyendo Argentina, Belice, Bolivia, Brasil, Chile, Colombia, El Salvador, Guatemala, Haití y México. Esta partición estratégica asegura que todas las operaciones relacionadas con estas regiones específicas se ejecuten exclusivamente en este nodo del sistema.

Por otro lado, el Servidor 2 está destinado a gestionar las regiones correspondientes a países ubicados en otros continentes, específicamente Europa, Asia, África y Oceanía. De manera análoga al primer servidor, este también cuenta con una restricción implementada que filtra y permite únicamente el registro de países correspondientes a estas regiones geográficas.

2.2.2. Fragmentación Vertical

La estrategia de fragmentación vertical implementada en este proyecto se basa en la separación de datos sensibles y no sensibles para abordar múltiples objetivos fundamentales: la seguridad, el rendimiento y la eficiencia en la administración de datos. Esta estrategia divide las tablas principales en fragmentos lógicos, trasladando la información confidencial a tablas separadas y distribuyendo estos fragmentos entre diferentes instancias de SQL Server.

Desde una perspectiva de seguridad, esta fragmentación permite aplicar controles de acceso específicos y medidas de protección más rigurosas para los datos sensibles, como encriptación y auditoría de accesos. Por ejemplo, los campos que contienen información personal identificable, como correos electrónicos, números de teléfono y estados de pedidos, se han aislado en tablas dedicadas. Esto reduce significativamente la exposición de datos confidenciales en caso de incidentes de seguridad, cumpliendo así con normativas de protección de datos como el GDPR.

Además, esta estrategia contribuye al rendimiento del sistema al optimizar las consultas en las tablas no sensibles. Dado que estas tablas ahora contienen menos columnas, las operaciones de lectura y escritura son más rápidas, lo que beneficia a las aplicaciones que no necesitan acceder a los datos sensibles de manera frecuente. Por otro lado, las consultas dirigidas a los fragmentos sensibles pueden gestionarse de forma más eficiente al estar en bases de datos separadas, permitiendo un enfoque personalizado para la gestión de la carga de trabajo.

Desde el punto de vista del diseño, la fragmentación vertical mejora la escalabilidad del sistema al distribuir la carga entre diferentes servidores. Por ejemplo, las tablas de datos sensibles pueden ubicarse en servidores con mayor capacidad de procesamiento o almacenamiento seguro, mientras que los datos generales permanecen en servidores estándar. Esta distribución permite una planificación más eficiente de los recursos y facilita la implementación de soluciones de alta disponibilidad y recuperación ante desastres.

2.3 PLAN DE REPLICACIÓN

2.3.1. Replicación Transaccional

En el contexto del proyecto de diseño e implementación de una base de datos distribuida avanzada, se ha seleccionado la replicación transaccional como una de las estrategias clave para garantizar la consistencia y la disponibilidad de los datos entre las distintas instancias del sistema. Esta decisión responde a las necesidades específicas del caso práctico, donde la empresa global de gestión de inventarios requiere sincronización casi en tiempo real y la capacidad de manejar grandes volúmenes de operaciones transaccionales.

La replicación transaccional es ideal para este proyecto porque permite que los cambios realizados en el publicador se propaguen a los suscriptores de manera incremental y con baja latencia. Esto asegura que las sucursales distribuidas de la empresa trabajen siempre con información actualizada, evitando discrepancias que podrían afectar la toma de decisiones o la operatividad. Además, dado que el sistema debe reflejar un entorno de alta disponibilidad, esta estrategia se alinea perfectamente con la necesidad de acceso confiable y en tiempo real a los datos.

La elección de la replicación transaccional también está justificada por las características del entorno operativo de la empresa. Dado que las bases de datos publicadoras manejan altos volúmenes de transacciones, esta estrategia es capaz de gestionar eficientemente la carga. Además, la replicación transaccional contribuye al diseño de un sistema robusto frente a fallos. Aunque los suscriptores son de solo lectura en la configuración predeterminada, esta estrategia asegura que los datos estén disponibles incluso si el publicador experimenta problemas, gracias a la sincronización continua de los cambios. En conjunto, estas características refuerzan la capacidad del sistema para operar en un entorno distribuido con alta disponibilidad y fiabilidad.

2.3.2. Replicación Combinacional

En el diseño de la base de datos distribuida avanzada para este proyecto, la replicación de combinación se ha seleccionado como una estrategia complementaria para abordar escenarios en los que los datos deben ser modificados en múltiples ubicaciones y luego sincronizados de manera eficiente. Esta estrategia resulta especialmente adecuada en entornos donde los datos son generados o actualizados tanto en el publicador como en los suscriptores, como en el caso de la empresa global de gestión de inventarios, que opera en múltiples regiones y requiere que estas trabajen con información unificada.

La replicación de combinación ofrece un enfoque flexible para la sincronización bidireccional de datos, permitiendo que las actualizaciones realizadas en cualquier nodo de la red se propaguen y consoliden en una base de datos coherente. En este proyecto, esta capacidad es fundamental para garantizar que los datos generados localmente en las sucursales (por ejemplo, inventarios regionales o pedidos específicos) se integren en el sistema global sin perder consistencia.

La replicación de combinación proporciona a la empresa una solución robusta y escalable para gestionar sus datos en un entorno distribuido. Al permitir actualizaciones locales y sincronización periódica con la base de datos central, la empresa obtiene flexibilidad operativa, una mayor disponibilidad del sistema y la capacidad de mantener la coherencia de los datos a nivel global.

En síntesis, la replicación de combinación es un pilar esencial en este proyecto, ya que facilita la integración de datos descentralizados, asegura la consistencia global y mejora la eficiencia operativa en una empresa con alcance internacional. Su implementación refuerza los objetivos de escalabilidad, alta disponibilidad y recuperación ante fallos, alineándose perfectamente con los requisitos del sistema distribuido diseñado.

2.4 ESTRATEGIA DE RESPALDO Y RECUPERACIÓN

La estrategia de respaldo y recuperación implementada en este proyecto tiene como objetivo garantizar la integridad y disponibilidad de los datos en un entorno de base de datos distribuida avanzada. En sistemas distribuidos, donde los datos se encuentran replicados o fragmentados en múltiples ubicaciones, es crucial contar con un esquema robusto que minimice el impacto de posibles fallos, ya sea a nivel de hardware, software o conexión de red. Esta estrategia se alinea con los requisitos de alta disponibilidad y recuperación ante desastres planteados para la empresa global de gestión de inventarios.

3

CONFIGURACIONES TÉCNICAS

3.1 CONFIGURACIÓN DE LAS INSTANCIAS DE SQL SERVER

La configuración de las instancias de SQL Server en el sistema distribuido siguió un proceso metódico para garantizar una comunicación efectiva entre las diferentes máquinas. El proceso comenzó con la verificación de direcciones IP mediante el comando `ipconfig`, ejecutado en cada máquina para identificar y establecer los parámetros de conexión necesarios entre los servidores.

En paralelo, se realizó la configuración del Firewall de Windows en cada máquina, un paso crítico para permitir la comunicación segura entre las instancias de SQL Server. Esto implicó la creación de reglas específicas tanto de entrada como de salida para el puerto 1433 (puerto predeterminado de SQL Server) y la habilitación de excepciones para SQL Server en el Firewall de Windows. También se configuraron reglas adicionales para el SQL Server Browser Service en el puerto UDP 1434, necesario para la resolución de instancias nombradas.

A continuación, se realizó una prueba exhaustiva de conectividad entre las máquinas utilizando el comando `ping`. Esta verificación fue crucial para asegurar que no existieran problemas de conectividad que pudieran comprometer la implementación del sistema distribuido. Las pruebas incluyeron la verificación de que los puertos configurados en el Firewall estuvieran correctamente abiertos y accesibles entre las máquinas.

En el aspecto de seguridad, se implementaron medidas específicas en cada instancia de SQL Server. Se accedió al apartado de “Inicio de sesión” dentro de las opciones de seguridad, donde se procedió a crear usuarios específicos para cada instancia. A estos usuarios se les otorgaron los permisos necesarios para gestionar datos y realizar replicación, asegurando así un acceso seguro y controlado al sistema.

Finalmente, se establecieron puntos de conexión dentro de la sección “Objetos del Servidor”. Esta configuración fue fundamental para permitir la interacción entre servidores, facilitando que las instancias de SQL Server pudieran replicar datos, realizar consultas distribuidas y mantener la integridad de todas las operaciones en el sistema. La combinación de estas configuraciones de red, firewall y seguridad creó un entorno robusto y seguro para la operación del sistema distribuido de SQL Server.

3.2 IMPLEMENTACIÓN DE LA FRAGMENTACIÓN

3.2.1. Fragmentación horizontal

En cuanto a la integración global, el sistema implementa una solución elegante para proporcionar una visión unificada de todas las regiones. Esto se logra mediante la creación de una vista global unificada, implementada a través de la instrucción UNION ALL. Esta vista cumple la función crucial de combinar los datos de la tabla Región provenientes de ambas instancias de SQL Server. La ventaja principal de esta implementación es que permite realizar consultas transparentes sobre todas las regiones disponibles, sin que sea necesario que los usuarios conozcan la ubicación física de los datos subyacentes.

3.2.2. Fragmentación vertical

El Servidor 1, designado para datos generales, contiene las columnas no sensibles de las tablas principales (Usuario, DetallePedido, Inventario), manteniendo únicamente la información básica y estructural necesaria para las operaciones estándar del sistema. Por otro lado, el Servidor 2 se dedica exclusivamente al almacenamiento de datos sensibles, incluyendo campos específicos como email y teléfono de la tabla Usuario, cantidad, precioUnitario y subtotal de DetallePedido, así como cantidad y ultimaActualización de la tabla Inventario.

En cuanto a la implementación técnica, se siguió un proceso metódico que comenzó con la migración de los datos sensibles. Las columnas sensibles fueron transferidas a nuevas tablas en una base de datos separada (DB_DatosSensibles) ubicada en el servidor 2. Después de completar esta migración, las columnas originales fueron eliminadas de las tablas en el servidor 1, asegurando así una separación efectiva de los datos.

Para facilitar un acceso transparente y unificado a la información, se implementó un sistema de vistas que combina los datos generales y sensibles de cada tabla fragmentada. Por ejemplo, la vista Usuario_Unificado combina las columnas no sensibles de la tabla Usuario del servidor 1 con las columnas sensibles almacenadas en el servidor 2. Esta integración se logró mediante una combinación LEFT JOIN, garantizando que todas

las filas de las tablas generales estén disponibles, incluso cuando no existan datos sensibles asociados.

3.3 CONFIGURACIÓN DE LA REPLICACIÓN

La configuración de la replicación en este proyecto se llevó a cabo con el objetivo de garantizar la consistencia de los datos, la alta disponibilidad y un acceso eficiente dentro de un entorno de base de datos distribuida. Dado que la empresa global de gestión de inventarios opera en diversas regiones geográficas, la replicación se implementó como una herramienta esencial para sincronizar los datos entre los diferentes nodos del sistema, minimizando los tiempos de inactividad y asegurando una operación fluida.

Para cumplir con los requisitos del sistema, se utilizaron dos enfoques principales de replicación: transaccional y combinacional. La replicación transaccional se configuró para sincronizar los datos en tiempo real entre las instancias de SQL Server. Este tipo de replicación resulta particularmente adecuado en escenarios con un alto volumen de transacciones, ya que permite que los cambios realizados en la base de datos publicadora (publicador) sean capturados y transferidos casi inmediatamente a las bases de datos suscriptoras (suscriptores).

El proceso de replicación transaccional comienza con la creación de una instantánea inicial de las tablas y esquemas publicados. Esto garantiza que todos los suscriptores partan de un estado consistente. Posteriormente, los cambios incrementales, como inserciones, actualizaciones o eliminaciones realizadas en el publicador, se transmiten a los suscriptores respetando el orden y los límites de las transacciones originales. Este enfoque no solo preserva la coherencia transaccional, sino que también asegura que los suscriptores reflejen de manera exacta el estado actual del publicador.

La implementación de la replicación transaccional en este proyecto utiliza los siguientes componentes de SQL Server:

- **Agente de instantáneas:** Este agente genera la instantánea inicial que contiene los datos y esquemas necesarios para que los suscriptores comiencen sincronizados. Este paso es esencial para establecer un punto de partida uniforme en el sistema distribuido.
- **Agente de registro del LOG:** Monitorea continuamente el registro de transacciones del publicador y transfiere las operaciones marcadas para replicación a la base de datos de distribución. Esto garantiza que cada cambio sea capturado y almacenado de manera segura para su posterior distribución.
- **Agente de distribución:** Se encarga de enviar los cambios incrementales desde el publicador hacia los suscriptores, permitiendo que estos se mantengan actualizados de manera continua o a intervalos programados, según las necesidades del sistema.

Por otro lado, se implementó la replicación combinacional para abordar la necesidad de integrar datos de múltiples regiones y permitir actualizaciones bidireccionales. Este método permite que los datos sean modi-

ficados tanto en el publicador como en los suscriptores, promoviendo una mayor flexibilidad operativa. Para evitar inconsistencias y conflictos entre los nodos, se configuraron políticas específicas de resolución de conflictos, las cuales priorizan la integridad lógica de los datos y minimizan posibles pérdidas de información. Este tipo de replicación es especialmente útil para consolidar datos regionales en un sistema centralizado, manteniendo la consistencia general.

El proceso de replicación de combinación utiliza agentes específicos para capturar cambios y resolver conflictos. Estos incluyen:

- Agente de combinación: Este componente monitorea y registra las modificaciones realizadas tanto en el publicador como en los suscriptores. Durante el proceso de sincronización, el agente aplica los cambios acumulados a todas las bases de datos involucradas, garantizando que cada nodo refleje un estado actualizado y consistente.
- Resolución de conflictos: Dado que los datos pueden modificarse simultáneamente en múltiples ubicaciones, la replicación de combinación incluye mecanismos avanzados para resolver conflictos. Esto asegura que las actualizaciones concurrentes se manejen de manera ordenada, siguiendo políticas predefinidas que priorizan ciertas operaciones o fuentes de datos.

3.4 CONFIGURACIÓN DE BACKUPS AUTOMÁTICOS

La configuración de los backups automáticos en este proyecto se estructuró en tres tipos de respaldos principales, cada uno con una frecuencia distinta y un propósito específico:

- Respaldo completo semanal: Este respaldo se realiza una vez a la semana y captura una copia completa de todas las bases de datos publicadas. Es fundamental para establecer un punto de recuperación base desde el cual se pueden aplicar respaldos diferenciales y de registros de transacciones. Este tipo de respaldo garantiza que siempre haya un punto de restauración reciente y confiable, permitiendo que el sistema pueda ser restaurado a su estado más reciente en caso de un desastre mayor.
- Respaldo diferencial diario: Los respaldos diferenciales se realizan de manera diaria y contienen solo los cambios realizados desde el último respaldo completo. Esta estrategia reduce la cantidad de espacio requerido para almacenar los backups y, al mismo tiempo, facilita una recuperación más rápida que si se dependiera únicamente de los respaldos completos. De este modo, se garantiza que las modificaciones diarias en las bases de datos se mantengan seguras y puedan ser restauradas de manera eficiente sin necesidad de restaurar todos los datos desde cero.
- Respaldo de registros de transacciones cada hora: Este tipo de respaldo captura todas las transacciones realizadas desde el último respaldo de registro de transacciones. Se realiza cada hora para garantizar que cualquier operación crítica, como inserciones, actualizaciones o eliminaciones, se registre de manera

precisa. En caso de fallos o interrupciones, este tipo de respaldo permite realizar una recuperación casi en tiempo real de las transacciones más recientes, minimizando la pérdida de datos.

La configuración de los respaldos automáticos se realizó utilizando las herramientas proporcionadas por SQL Server Management Studio (SSMS). En particular, se utilizó el Agente SQL Server para programar los respaldos de acuerdo con las políticas definidas y asegurar que estos se ejecutaran de forma automática en los horarios establecidos. Además, se configuraron alertas para notificar a los administradores de cualquier fallo o inconsistencia en el proceso de respaldos.

En caso de una falla en alguno de los nodos o de una interrupción en el servicio, los backups automáticos juegan un papel crucial en la recuperación del sistema. Gracias a la combinación de respaldos completos, diferenciales y de registros de transacciones, el sistema puede restaurarse a su estado más reciente con la menor pérdida de datos posible. Esto se logra mediante un proceso de restauración secuencial en el que se comienza con el respaldo completo más reciente, seguido de la aplicación de los respaldos diferenciales y finalmente, los registros de transacciones, asegurando una recuperación fiel del sistema hasta el último cambio realizado.

En las figuras 3, 4 y 5 se muestra el código de los distintos backups y el proceso de cómo se programan los jobs para ejecutar los backups semanal, diario y a cada hora.

3.5 CONFIGURACIÓN DE NIVELES DE AISLAMIENTO

SQL Server ofrece varios niveles de aislamiento que determinan cómo las transacciones pueden interactuar entre sí. Cada nivel de aislamiento define un grado diferente de visibilidad de los datos entre transacciones concurrentes, afectando la cantidad de bloqueos y la consistencia de las lecturas. Para el diseño e implementación de la base de datos distribuida en este proyecto, se consideraron los siguientes niveles de aislamiento:

- Read Uncommitted: Este es el nivel de aislamiento más bajo y permite que las transacciones lean datos que aún no han sido confirmados por otras transacciones.
- Read Committed: En este nivel, una transacción solo puede leer datos que hayan sido confirmados, evitando las lecturas sucias.
- Repeatable Read: Este nivel de aislamiento garantiza que, una vez que una transacción lee un valor, ese valor no cambiará durante el curso de la transacción, evitando las lecturas no repetibles.

En este proyecto, se adoptaron niveles de aislamiento ajustados a las necesidades específicas de las transacciones en el sistema distribuido, buscando un equilibrio entre rendimiento y consistencia:

- En las operaciones de lectura intensiva y aquellas donde no se requería la mayor precisión transaccional, se utilizó el nivel Read Committed para evitar lecturas sucias mientras se mantenía una alta disponibilidad y rendimiento.

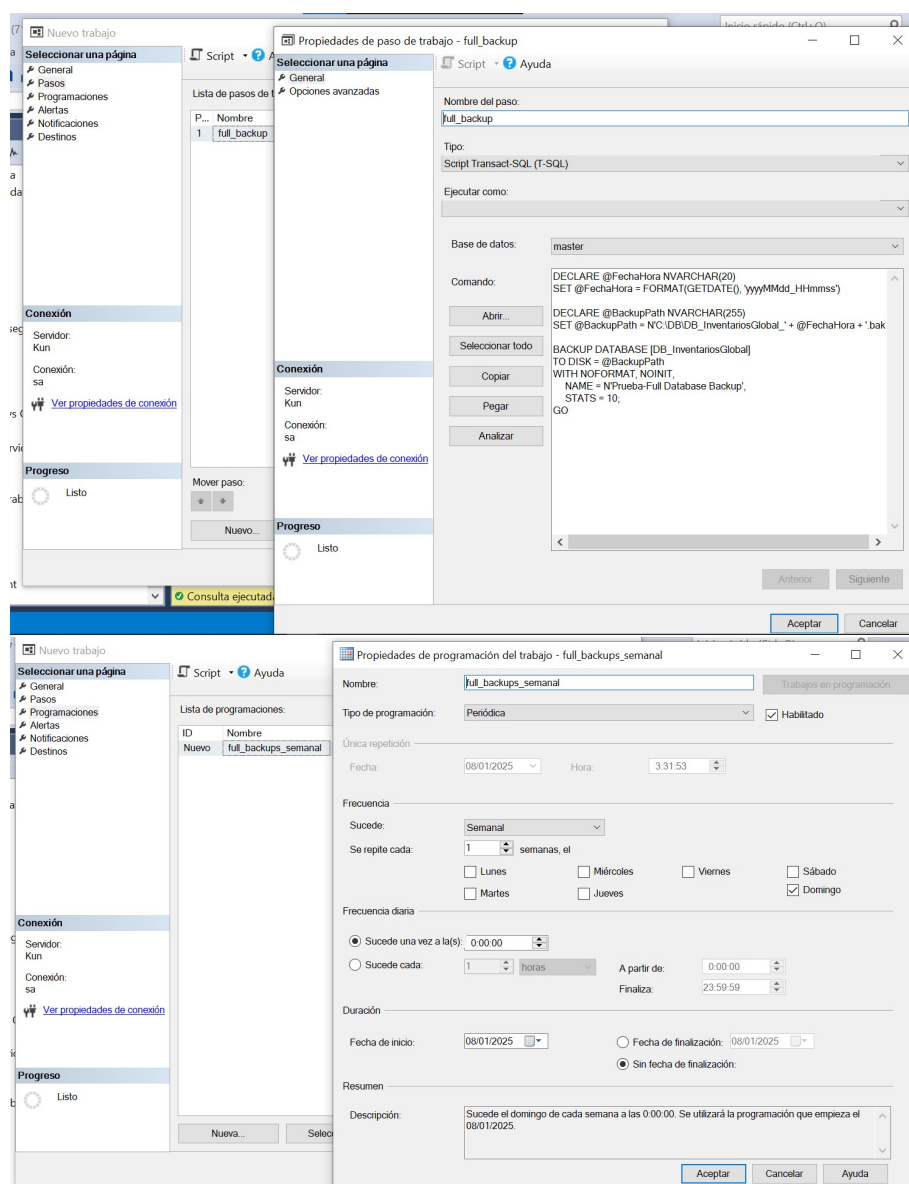


Figura 3: Programación del FullBackup.

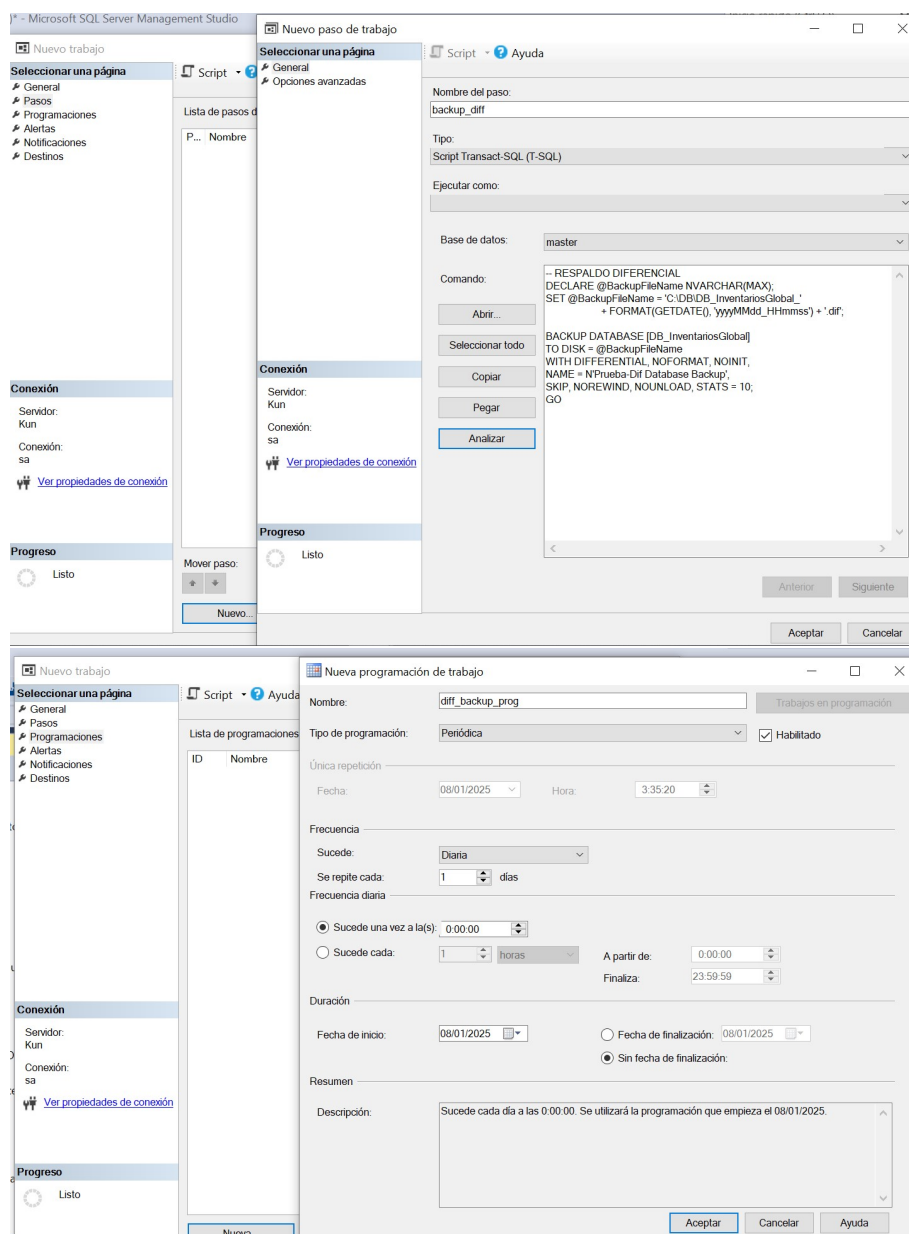


Figura 4: Programación del Backup Diferencial.

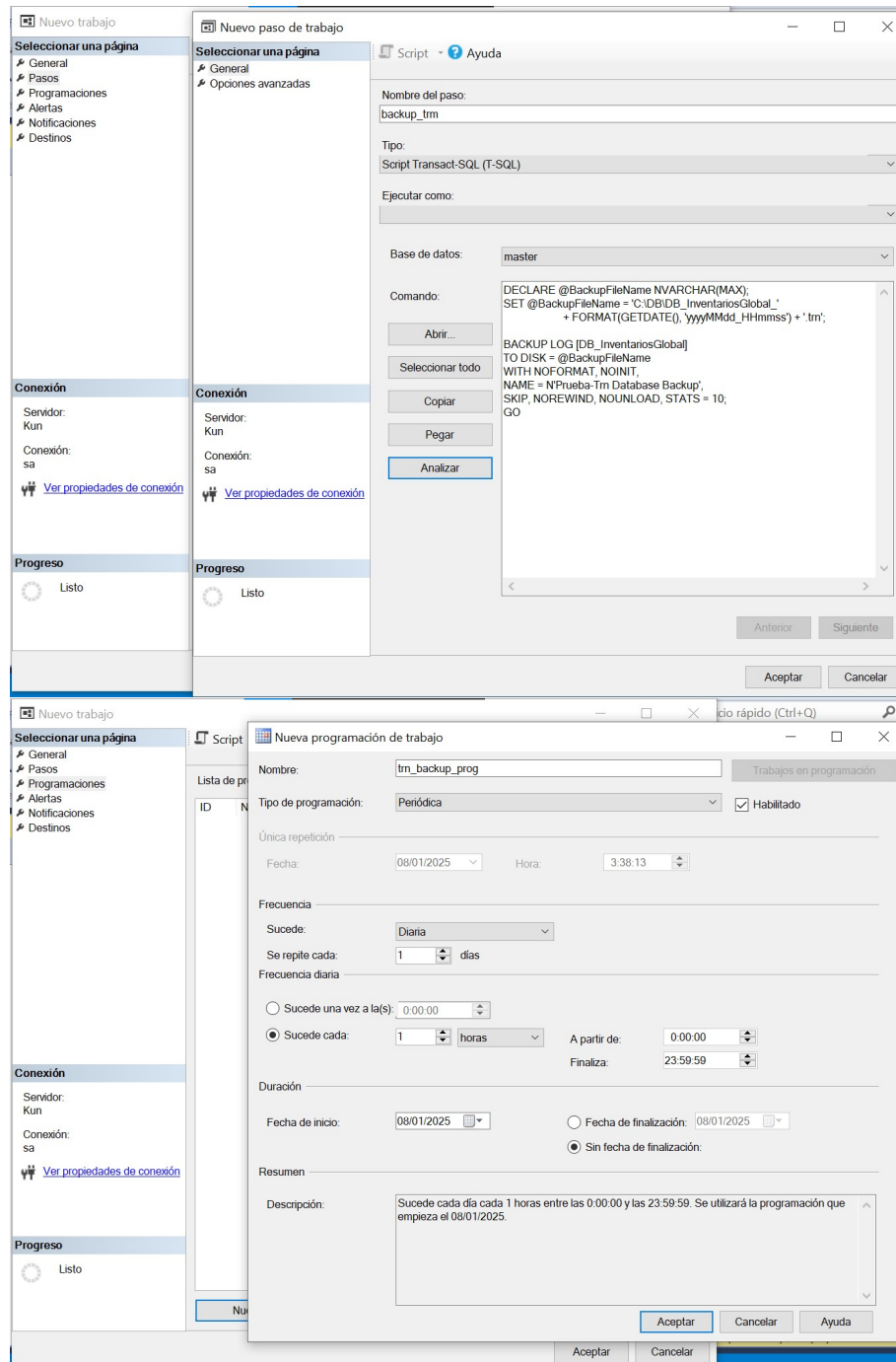


Figura 5: Programación del Backup Transaccional.

- En las operaciones críticas de escritura, como las actualizaciones de inventarios y transacciones financieras, se empleó el nivel Serializable para garantizar que las transacciones fueran aisladas y no interfieran entre sí, asegurando que no ocurrieran inconsistencias como las lecturas fantasmas.
- En los casos donde se necesitaba realizar lecturas repetibles de registros, como en las consultas de datos históricos o análisis que involucraban múltiples operaciones de lectura, se utilizó el nivel Repeatable Read para asegurar que los datos no fueran modificados durante la transacción.

4

PRUEBAS Y RESULTADOS

4.1 PRUEBAS DE FRAGMENTACIÓN

4.1.1. Fragmentación Horizontal

Para las pruebas de la fragmentación horizontal, la Figura 6 ilustra cómo se poblaron las tablas en las diferentes bases de datos distribuidas, fragmentándolas de acuerdo con su región correspondiente. Para validar su correcto funcionamiento, la Figura 7 muestra el total de elementos resultantes tras la creación de una vista que unifica los fragmentos de datos generados previamente.

regionID	nombre	pais	zonaHoraria
1	Ciudad Autónoma de Buenos Aires	Argentina	UTC-3
2	La Plata	Argentina	UTC-3
3	Mar del Plata	Argentina	UTC-3
4	Bahía Blanca	Argentina	UTC-3
5	Quilmes	Argentina	UTC-3
6	Lanús	Argentina	UTC-3
7	Lomas de Zamora	Argentina	UTC-3
8	Tandil	Argentina	UTC-3
9	San Isidro	Argentina	UTC-3

regionID	nombre	pais	zonaHoraria
281	Sydney	Australia	UTC+10
282	Newcastle	Australia	UTC+10
283	Wollongong	Australia	UTC+10
284	Central Coast	Australia	UTC+10
285	Wagga Wagga	Australia	UTC+10
286	Albury	Australia	UTC+10
287	Port Macquarie	Australia	UTC+10
288	Tamworth	Australia	UTC+10

Figura 6: Fragmentación Horizontal.

4.1.2. Fragmentación Vertical

En cuanto a las pruebas de la fragmentación vertical, la Figura 8 muestra cómo se realizó la separación de la tabla Usuario, dividiendo los datos sensibles de la tabla original y almacenándolos en una base de datos

	TotalRegiones
1	10055

Consulta ejecut... KUN (16.0 RTM) sa (59) DB_InventariosGlobal 00:00:00 1 filas

Figura 7: Vista Horizontal.

diferente. Por su parte, la Figura 9 presenta el resultado de crear una vista que unifica todos los fragmentos de datos previamente separados, permitiendo un acceso integrado y coherente a la información.

	usuarioID	nombre	apellido	regionID	email	telefono
1	100101255	Hunter	Kase	5424	hkasebr@bloomberg.com	491-818-4450
2	100109417	Corella	Tuhy	1054	ctuhy1h@indiegogo.com	496-453-0964
3	100198154	Orellie	Morgans	8406	omorgansb9@list-mana...	374-467-6147
4	100272978	Reamonn	Faulo	223	rfaulo66@slate.com	419-694-3607
5	100794297	Kaycee	Mallon	4381	kmallonak@histats.com	692-442-4526
6	100852164	Kristen	Sager	4294	ksager85@fastcompan...	177-230-5961
7	100979802	Tully	McTeer	1652	tmcteeri3@booking.com	238-379-6733
8	101851372	Karee	Lefley	2679	klefleyu@devhub.com	165-883-8465

Consulta ejecutada correctamente. KUN (16.0 RTM) sa (53) DB_InventariosGlobal 00:00:01 11.484 filas

Figura 8: Fragmentacion Vertical.

	usuarioID	nombre	apellido	regionID	email	telefono
1	100101255	Hunter	Kase	5424	hkasebr@bloomberg.com	491-818-4450
2	100109417	Corella	Tuhy	1054	ctuhy1h@indiegogo.com	496-453-0964
3	100198154	Orellie	Morgans	8406	omorgansb9@list-manage.com	374-467-6147
4	100272978	Reamonn	Faulo	223	rfaulo66@slate.com	419-694-3607
5	100794297	Kaycee	Mallon	4381	kmallonak@histats.com	692-442-4526
6	100852164	Kristen	Sager	4294	ksager85@fastcompany.com	177-230-5961
7	100979802	Tully	McTeer	1652	tmcteeri3@booking.com	238-379-6733
8	101851372	Karee	Lefley	2679	klefleyu@devhub.com	165-883-8465
9	101855488	Willette	Riglesford	4363	wriglesforda2@house.gov	314-293-0106
10	102791865	Yank	Orr	8257	yorr74@ucsd.edu	807-323-2389
11	103325649	Cathee	Imbrey	5356	cimbrey9w@princeton.edu	730-162-1458
12	104195425	Damiano	Rollinson	1093	drollinson2k@netscape.com	197-699-1400
13	105276464	Kelbee	Kemp	4318	kkemp8t@sun.com	227-588-6804
14	105510466	Brien	Gaul	5419	bgaubm@networkadvertising.org	473-420-5065
15	105780684	Tania	Keltridge	106	tkeltridge2x@redcross.org	678-190-3792
16	106293746	Ulrica	Bordone	1072	ubordone1z@artisteer.com	360-856-8932
17	106806529	Frederich	Merchant	2758	fmerchant1l@kickstarter.com	977-512-6402

Consulta ejecutada correctamente. KUN (16.0 RTM) sa (70) DB_InventariosGlobal 00:00:00 1.996 filas

Figura 9: Vista Vertical.

4.2 PRUEBAS DE RECUPERACIÓN ANTE FALLOS

La recuperación ante fallos se garantiza mediante la configuración de trabajos automáticos para la generación de respaldos de la base de datos. Estos respaldos se realizan con una frecuencia programada: semanalmente (respaldo completo), diariamente (respaldo diferencial) y cada hora (respaldo transaccional). En caso de fallo, se procede a realizar una restauración, ya sea completa, diferencial o transaccional, dependiendo de

la naturaleza y el alcance de la recuperación requerida. Esta estrategia asegura la mínima pérdida de datos y la pronta restauración del sistema. La restauración se puede observar de una mejor manera en la Figura 10.

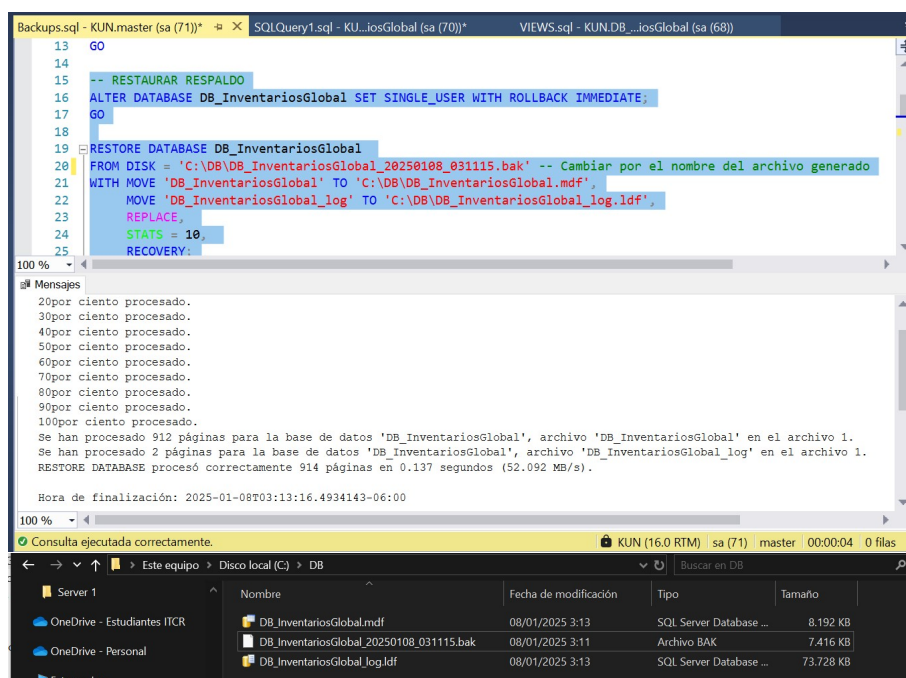


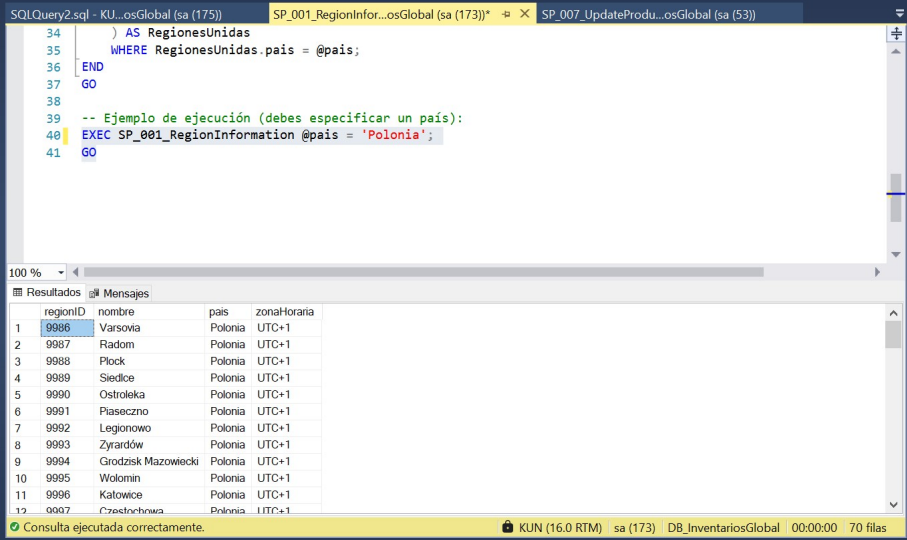
Figura 10: Restauracion de backup.

4.3 RESULTADOS DE CONSULTAS DISTRIBUIDAS

En las Figuras 11 y 12 se presentan dos escenarios clave en la interacción con bases de datos distribuidas mediante el uso de procedimientos almacenados. Estas herramientas permiten realizar operaciones eficientes y seguras en sistemas distribuidos, facilitando la consulta y actualización de datos en diferentes instancias de bases de datos.

La Figura 11 ilustra el procedimiento almacenado SP_ooi_RegionInformation, diseñado para consultar información sobre las regiones de un país específico. Este procedimiento se ejecuta mediante un parámetro de entrada que especifica el país deseado, en este caso "Polonia". Como resultado, se obtiene un conjunto de datos que incluye el ID de la región, su nombre, el país al que pertenece y su zona horaria. Este tipo de consulta es esencial en sistemas distribuidos donde los datos están segmentados por regiones o países, ya que permite recuperar información relevante desde diferentes servidores o instancias.

Por otro lado, la Figura 12 muestra el procedimiento almacenado SP_oo7_UpdateProduct, que se utiliza para actualizar el precio y la categoría de un producto en la base de datos. Este procedimiento recibe como parámetros el ID del producto, el nuevo precio y la nueva categoría. En caso de que el producto exista en la base de datos, el procedimiento almacena los cambios y devuelve un mensaje de éxito. De no encontrar el



```

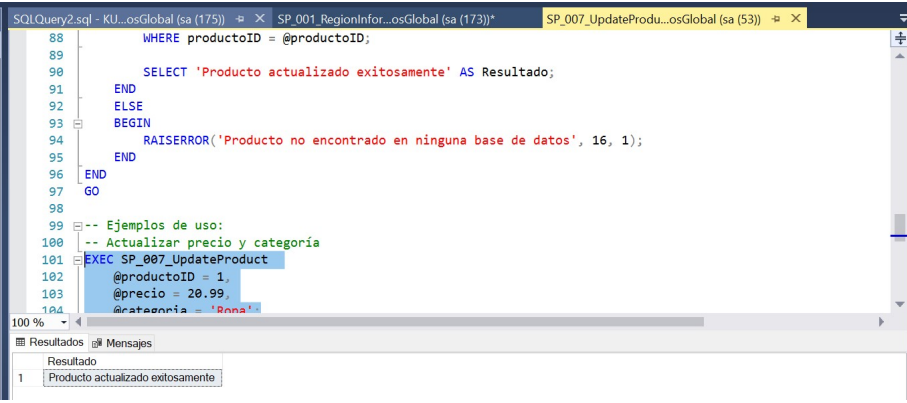
34      ) AS RegionesUnidas
35      WHERE RegionesUnidas.pais = @pais;
36  END
37  GO
38
39  -- Ejemplo de ejecución (debes especificar un país):
40  EXEC SP_001_RegionInformation @pais = 'Polonia';
41  GO

```

regionID	nombre	pais	zonaHoraria
1	9986	Varsovia	Polonia UTC+1
2	9987	Radom	Polonia UTC+1
3	9988	Plock	Polonia UTC+1
4	9989	Siedlce	Polonia UTC+1
5	9990	Ostroleka	Polonia UTC+1
6	9991	Piaseczno	Polonia UTC+1
7	9992	Legionowo	Polonia UTC+1
8	9993	Zyrardow	Polonia UTC+1
9	9994	Grodzisk Mazowiecki	Polonia UTC+1
10	9995	Wolomin	Polonia UTC+1
11	9996	Katowice	Polonia UTC+1
12	9997	Czestochowa	Polonia UTC+1

Consulta ejecutada correctamente.

Figura 11: Consulta de Regiones de Polonia por unión de datos de ambos servidores.



```

88      WHERE productoID = @productoID;
89
90      SELECT 'Producto actualizado exitosamente' AS Resultado;
91  END
92  ELSE
93  BEGIN
94      RAISERROR('Producto no encontrado en ninguna base de datos', 16, 1);
95  END
96  END
97  GO
98
99  -- Ejemplos de uso:
100  -- Actualizar precio y categoría
101  EXEC SP_007_UpdateProduct
102  @productoID = 1,
103  @precio = 20.99,
104  @categoria = 'Bona';

```

Resultado	
1	Producto actualizado exitosamente

Figura 12: Actualización de Datos del Segundo Servidor mediante el primer servidor.

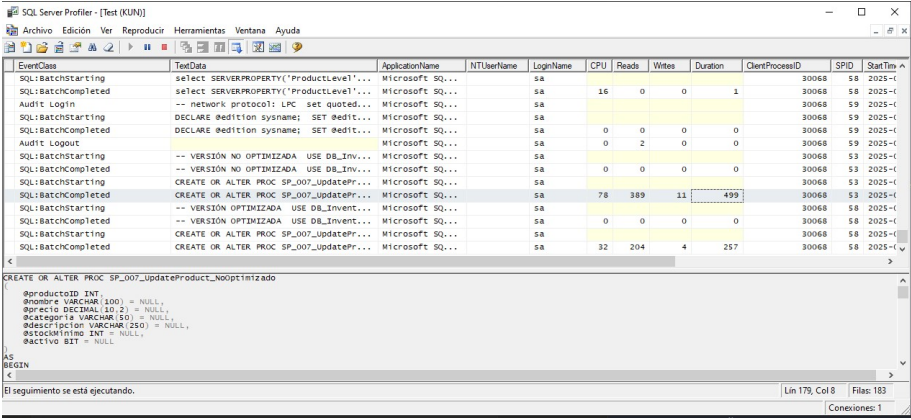
producto, genera un error personalizado, indicando que el registro no está disponible. Este enfoque asegura una correcta manipulación de los datos en sistemas distribuidos, donde las actualizaciones deben ser precisas y garantizar la integridad de los registros.

5

PRUEBAS DE RENDIMIENTO

En esta sección se presentan los resultados de las pruebas de rendimiento realizadas sobre diversos procedimientos almacenados, destacando las diferencias entre implementaciones optimizadas y no optimizadas.

En la Figura 13 se observa la ejecución de un procedimiento almacenado en su estado no optimizado, con un uso de 78 unidades de CPU, 389 lecturas, 11 escrituras y una duración total de 499 ms. Este alto consumo de recursos se debe a la presencia de numerosas validaciones redundantes y anidadas, así como a la falta de un manejo adecuado de la concurrencia, lo que genera una ejecución ineficiente y un mayor tiempo de respuesta.



EventClass	EventData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Duration	ClientProcessID	SPID	StartTime
SQL:BatchStarting	select SERVERPROPERTY('ProductLevel')...	Microsoft SQ...	sa	sa					30068	58	2025-11-01 10:00:00
SQL:BatchCompleted	select SERVERPROPERTY('ProductLevel')...	Microsoft SQ...	sa	sa	16	0	0	1	30068	58	2025-11-01 10:00:00
Audit:Login	-- network protocol: LPC set quoted...	Microsoft SQ...	sa	sa					30068	59	2025-11-01 10:00:00
SQL:BatchStarting	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa	sa					30068	59	2025-11-01 10:00:00
SQL:BatchCompleted	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa	sa	0	0	0	0	30068	59	2025-11-01 10:00:00
Audit:Logout		Microsoft SQ...	sa	sa	0	2	0	0	30068	59	2025-11-01 10:00:00
SQL:BatchStarting	-- VERSIÓN NO OPTIMIZADA USE DB_Inven...	Microsoft SQ...	sa	sa					30068	53	2025-11-01 10:00:00
SQL:BatchCompleted	-- VERSIÓN NO OPTIMIZADA USE DB_Inven...	Microsoft SQ...	sa	sa	0	0	0	0	30068	53	2025-11-01 10:00:00
SQL:BatchStarting	CREATE OR ALTER PROC SP_007_UpdatePr...	Microsoft SQ...	sa	sa					30068	53	2025-11-01 10:00:00
SQL:BatchCompleted	CREATE OR ALTER PROC SP_007_UpdatePr...	Microsoft SQ...	sa	sa	78	389	11	499	30068	53	2025-11-01 10:00:00
SQL:BatchStarting	-- VERSIÓN OPTIMIZADA USE DB_Invent...	Microsoft SQ...	sa	sa					30068	58	2025-11-01 10:00:00
SQL:BatchCompleted	-- VERSIÓN OPTIMIZADA USE DB_Invent...	Microsoft SQ...	sa	sa	0	0	0	0	30068	58	2025-11-01 10:00:00
SQL:BatchStarting	CREATE OR ALTER PROC SP_007_UpdatePr...	Microsoft SQ...	sa	sa					30068	58	2025-11-01 10:00:00
SQL:BatchCompleted	CREATE OR ALTER PROC SP_007_UpdatePr...	Microsoft SQ...	sa	sa	32	204	4	257	30068	58	2025-11-01 10:00:00

```

CREATE OR ALTER PROC SP_007_UpdateProduct_NoOptimizado
AS
BEGIN
    @productID INT
    @nombre VARCHAR(100) = NULL,
    @precio DECIMAL(10,2) = NULL,
    @categoria VARCHAR(50) = NULL,
    @descripcion VARCHAR(250) = NULL,
    @stockActual INT = NULL,
    @activo BIT = NULL
END

```

Figura 13: Código de procedimiento almacenado redundante.

Por el contrario, la Figura 14 ilustra el mismo procedimiento almacenado, pero optimizado. En este caso, los resultados muestran un consumo reducido de 32 unidades de CPU, 204 lecturas, 4 escrituras y una duración total de 257 ms. Esta mejora significativa se logró mediante la agrupación de validaciones en bloques más eficientes y la implementación de técnicas de optimización para el manejo de la concurrencia. La comparación entre ambas ejecuciones demuestra claramente la efectividad de la optimización en términos de rendimiento y utilización de recursos.

Finalmente, en la Figura 15 se presenta la ejecución de un procedimiento almacenado que realiza una consulta compleja sobre la tabla Inventario. En este caso, el consumo de recursos muestra 109 unidades de CPU, 61 lecturas, 0 escrituras, y una duración total de 35,393 ms. Este tiempo de ejecución prolongado se debe a que el procedimiento estuvo en espera de la finalización de una transacción bloqueada, lo que pone de manifiesto

EventClass	TextData	ApplicationName	NTUserName	LogName	CPU	Reads	Writes	Duration	ClientProcessID	SPID	StartTime
SQL:BatchStarting	select SERVERPROPERTY('ProductLevel')...	Microsoft SQ...	sa		0	0	0	0			
SQL:BatchCompleted	select SERVERPROPERTY('ProductLevel')...	Microsoft SQ...	sa		16	0	0	1			
Audit: Login	-- network protocol: LPC set quoted...	Microsoft SQ...	sa								
SQL:BatchStarting	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa								
SQL:BatchCompleted	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa		0	0	0	0			
Audit: Logout											
SQL:BatchStarting	-- VERSION NO OPTIMIZADA USE DB_Inven...	Microsoft SQ...	sa		0	2	0	0			
SQL:BatchCompleted	-- VERSION NO OPTIMIZADA USE DB_Inven...	Microsoft SQ...	sa		0	0	0	0			
SQL:BatchStarting	CREATE OR ALTER PROC SP_007_UpdatePr...	Microsoft SQ...	sa								
SQL:BatchCompleted	CREATE OR ALTER PROC SP_007_UpdatePr...	Microsoft SQ...	sa		78	389	11	499			
SQL:BatchStarting	-- VERSION OPTIMIZADA USE DB_Inven...	Microsoft SQ...	sa								
SQL:BatchCompleted	-- VERSION OPTIMIZADA USE DB_Inven...	Microsoft SQ...	sa		0	0	0	0			
SQL:BatchStarting	CREATE OR ALTER PROC SP_007_UpdatePr...	Microsoft SQ...	sa								
SQL:BatchCompleted	CREATE OR ALTER PROC SP_007_UpdatePr...	Microsoft SQ...	sa		32	204	4	257			

```

CREATE OR ALTER PROC SP_007_UpdateProduct_optimizado
AS
BEGIN
    @productoID INT,
    @nombre VARCHAR(100) = NULL,
    @precio DECIMAL(10,2) = NULL,
    @descripcion VARCHAR(250) = NULL,
    @activo BIT = NULL
END

```

El seguimiento se está ejecutando. Lin 183, Col 9 Filas: 183 Conexiones: 1

Figura 14: Código de procedimiento almacenado optimizado.

la importancia de manejar correctamente la concurrencia y las dependencias entre transacciones para evitar demoras significativas en el rendimiento del sistema.

Estas pruebas reflejan la relevancia de implementar buenas prácticas de diseño y optimización en los procedimientos almacenados, tanto para reducir los tiempos de ejecución como para mejorar el uso de los recursos disponibles en el sistema.

EventClass	TextData	ApplicationName	NTUserName	LogName	CPU	Reads	Writes	Duration	ClientProcessID	SPID	StartTime	EndTime	BatchData
SQL:BatchCompleted	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa		0	0	0	0					
SQL:BatchStarting	SET ROWCOUNT 0 SET TEXTSIZE 21474836...	Microsoft SQ...	sa		0	0	0	2					
SQL:BatchCompleted	SET ROWCOUNT 0 SET TEXTSIZE 21474836...	Microsoft SQ...	sa		0	0	0	2					
SQL:BatchStarting	select SERVERPROPERTY('ProductLevel')...	Microsoft SQ...	sa		0	0	0	2					
SQL:BatchCompleted	select SERVERPROPERTY('ProductLevel')...	Microsoft SQ...	sa		0	0	0	2					
Audit: Login	-- network protocol: LPC set quoted...	Microsoft SQ...	sa										
SQL:BatchStarting	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa										
SQL:BatchCompleted	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa		0	0	0	0					
Audit: Logout													
SQL:BatchStarting	USE DB_InventarioGlobal;	Microsoft SQ...	sa		0	2	0	0					
SQL:BatchCompleted	USE DB_InventarioGlobal;	Microsoft SQ...	sa		0	0	0	0					
SQL:BatchStarting	CREATE OR ALTER PROC SP_003_UserInfo...	Microsoft SQ...	sa		0	0	0	0					
SQL:BatchCompleted	CREATE OR ALTER PROC SP_003_UserInfo...	Microsoft SQ...	sa		15	272	1	310					
SQL:BatchStarting	-- Ejemplo de ejecución: EXEC SP_...	Microsoft SQ...	sa										
SQL:BatchCompleted	-- Ejemplo de ejecución: EXEC SP_...	Microsoft SQ...	sa		47	104	0	408					
SQL:BatchStarting	-- network protocol: LPC set quoted...	Microsoft SQ...	sa										
SQL:BatchCompleted	-- network protocol: LPC set quoted...	Microsoft SQ...	sa										
SQL:BatchStarting	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa		0	0	0	0					
SQL:BatchCompleted	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa		0	0	0	0					
SQL:BatchStarting	SET ROWCOUNT 0 SET TEXTSIZE 21474836...	Microsoft SQ...	sa		0	0	0	0					
SQL:BatchCompleted	SET ROWCOUNT 0 SET TEXTSIZE 21474836...	Microsoft SQ...	sa		0	0	0	0					
SQL:BatchStarting	select SERVERPROPERTY('ProductLevel')...	Microsoft SQ...	sa		0	0	0	2					
SQL:BatchCompleted	select SERVERPROPERTY('ProductLevel')...	Microsoft SQ...	sa		0	0	0	2					
Audit: Login	-- network protocol: LPC set quoted...	Microsoft SQ...	sa										
SQL:BatchStarting	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa										
SQL:BatchCompleted	DECLARE @edition sysname; SET @edit...	Microsoft SQ...	sa		16	0	0	0					
Audit: Logout													
SQL:BatchStarting	USE DB_InventarioGlobal;	Microsoft SQ...	sa		16	2	0	0					
SQL:BatchCompleted	USE DB_InventarioGlobal;	Microsoft SQ...	sa		0	0	0	0					
SQL:BatchStarting	CREATE OR ALTER PROC SP_004_Inventor...	Microsoft SQ...	sa		0	0	0	0					
SQL:BatchCompleted	CREATE OR ALTER PROC SP_004_Inventor...	Microsoft SQ...	sa		0	353	0	244					
SQL:BatchStarting	-- Ejemplo de ejecución: EXEC SP_...	Microsoft SQ...	sa										
SQL:BatchCompleted	-- Ejemplo de ejecución: EXEC SP_...	Microsoft SQ...	sa		109	81	0	3535					

Figura 15: Mayor tiempo de espera ante una transacción ejecutada en el otro servidor.

CONCLUSIONES Y RECOMENDACIONES

6.1 CONCLUSIONES

En el desarrollo del proyecto, se abordaron y resolvieron varios aspectos técnicos clave relacionados con la administración y rendimiento de bases de datos distribuidas. Se logró implementar estrategias de fragmentación horizontal y vertical, demostrando su utilidad para distribuir los datos según regiones o separar información sensible, asegurando tanto la eficiencia como la seguridad. Las pruebas realizadas evidenciaron que estas

técnicas contribuyen significativamente a optimizar las consultas y a simplificar la gestión de los datos.

Por otro lado, la implementación de respaldos automáticos y estrategias de recuperación ante fallos proporcionó un mecanismo robusto para garantizar la continuidad del servicio ante posibles eventualidades. La configuración de niveles de aislamiento y el uso de índices particionados contribuyeron a minimizar problemas de concurrencia y a mejorar el rendimiento general de las consultas.

Las pruebas de rendimiento destacaron la importancia de optimizar procedimientos almacenados. La comparación entre procedimientos no optimizados y optimizados evidenció mejoras significativas en términos de consumo de CPU, lecturas, escrituras y tiempo de ejecución, reforzando la necesidad de prácticas eficientes de desarrollo.

6.2 RECOMENDACIONES

Se recomienda monitorear de manera constante tanto la replicación transaccional como la combinacional. Aunque ambas estrategias demostraron ser eficaces, la implementación de herramientas de monitoreo permitirá identificar posibles fallos o retrasos en la sincronización de datos. De este modo, se garantizará que los datos sean consistentes en todo momento, lo cual es esencial para mantener la integridad del sistema distribuido.

En relación con la estrategia de respaldo, es crucial seguir manteniendo actualizados los planes de respaldo y recuperación ante fallos. Verificar la integridad de los backups periódicamente y simular escenarios de recuperación ayudará a garantizar que los respaldos sean efectivos en situaciones de emergencia. Esto es esencial para asegurar la continuidad del servicio en caso de incidentes.

El uso de índices particionados ha demostrado ser una estrategia útil para mejorar el rendimiento en este proyecto, por lo que se recomienda considerar su implementación en otras tablas con grandes volúmenes de datos. Este enfoque ayudará a optimizar las consultas, mejorando la eficiencia y reduciendo los tiempos de respuesta, especialmente en sistemas que manejan grandes cantidades de información.

Por último, es recomendable realizar pruebas de estrés y carga en entornos de producción para simular escenarios de alto tráfico y grandes volúmenes de datos. Estas pruebas permitirán identificar limitaciones en el rendimiento y corregir posibles problemas antes de que afecten a los usuarios finales.