PONTIFICIA UNIVERSIDAD CATOLICA DEL ECUADOR
SEDE ESMERALDAS

ADMINISTRACION DE BASE DE DATOS

TEMA:

INFORME DE ANÁLISIS DE IMPLEMENTACIÓN DE BASE DE DATOS DISTRIBUIDA

AUTORES:

JEFFERSON MANUEL CUERVO CASTILLO
ANDRES ALEJANDRO CARVAJAL ARIAS
RODDY GERALD ALMEIDA SAQUISARI

TUTOR:

Prof. ESTEBAN JIMENEZ

ESMERALDAS – ECUADOR

2024 - 2025

RESUMEN EJECUTIVO

El presente documento constituye un análisis exhaustivo de un sistema de Base de Datos Distribuida desarrollado para una compañía vinícola española con operaciones en cuatro delegaciones territoriales. El sistema implementa una arquitectura distribuida mediante fragmentación horizontal basada en criterios geográficos, utilizando Oracle Database como gestor de base de datos.

El análisis abarca la revisión completa de la arquitectura, la identificación de estrategias de fragmentación y replicación, la evaluación de mecanismos de integridad distribuida, y la ejecución de consultas que integran datos de múltiples nodos. Se ha verificado la correcta implementación de conceptos fundamentales como autonomía local, transparencia de distribución y consistencia de datos, identificando además áreas de mejora en aspectos como la gestión de transacciones distribuidas y la escalabilidad del sistema.

INTRODUCCIÓN

Contexto del Proyecto

Las organizaciones modernas con presencia geográfica dispersa enfrentan desafíos significativos en la gestión de información distribuida. El caso de estudio analizado corresponde a una compañía vinícola española que opera a través de cuatro delegaciones regionales, cada una responsable de atender clientes en comunidades autónomas específicas. Esta estructura organizacional natural sugiere la implementación de una Base de Datos Distribuida como solución óptima para gestionar eficientemente la información mientras se respeta la autonomía operativa de cada delegación.

Objetivos del Análisis

El presente análisis tiene como propósito principal examinar en profundidad la arquitectura y funcionamiento de un sistema de base de datos distribuida real, evaluando la aplicación práctica de conceptos teóricos fundamentales. Los objetivos específicos incluyen la identificación del esquema de fragmentación implementado, la comprensión de los mecanismos que garantizan la integridad y consistencia de datos distribuidos, la evaluación de estrategias de transparencia y autonomía, y la verificación práctica del funcionamiento mediante la ejecución de consultas distribuidas.

Alcance del Documento

Este documento presenta un estudio completo que abarca desde los fundamentos teóricos de bases de datos distribuidas hasta la implementación práctica de consultas complejas que integran datos de múltiples nodos. Se incluyen instrucciones detalladas para la replicación del entorno de pruebas, especificaciones sobre las evidencias que deben capturarse durante la ejecución, y un análisis crítico que identifica fortalezas y limitaciones del diseño.

MARCO TEÓRICO

Bases de Datos Distribuidas

Una Base de Datos Distribuida consiste en una colección de múltiples bases de datos lógicamente interrelacionadas, distribuidas físicamente a través de una red de computadoras. A diferencia de las bases de datos centralizadas tradicionales, donde toda la información reside en un único servidor, los sistemas distribuidos dispersan los datos en múltiples localizaciones geográficas manteniendo la ilusión de un sistema único y coherente para los usuarios finales.

El fundamento teórico de estos sistemas se basa en el principio de que la distribución física de datos debe ser transparente para las aplicaciones y usuarios, quienes interactúan con el sistema como si fuera centralizado. Esta transparencia se logra mediante capas de abstracción que ocultan la complejidad de la distribución, resolución de consultas distribuidas y coordinación entre nodos.

Principios Fundamentales

El diseño de bases de datos distribuidas se rige por doce reglas fundamentales establecidas por C.J. Date, entre las cuales destacan la autonomía local, ausencia de dependencia de un sitio central, operación continua, independencia de localización, independencia de fragmentación, independencia de replicación y procesamiento distribuido de consultas. Estos principios garantizan que el sistema distribuido mantenga las propiedades ACID de las bases de datos tradicionales mientras aprovecha las ventajas de la distribución geográfica.

La autonomía local implica que cada nodo debe poder operar independientemente, gestionando sus propios datos sin requerir comunicación constante con otros nodos. La transparencia de localización significa que los usuarios no necesitan conocer dónde residen físicamente los datos para acceder a ellos. La transparencia de fragmentación permite que los usuarios trabajen con vistas globales sin preocuparse por cómo se han dividido las tablas entre nodos.

Fragmentación de Datos

La fragmentación constituye el proceso de dividir una relación global en múltiples fragmentos que se almacenan en diferentes nodos de la red. Existen tres tipos principales de fragmentación. La fragmentación horizontal divide una tabla en subconjuntos de filas basándose en predicados de selección, siendo útil cuando diferentes localizaciones trabajan con diferentes subconjuntos de datos. La fragmentación vertical divide una tabla en subconjuntos de columnas, apropiada cuando diferentes aplicaciones requieren diferentes atributos. La fragmentación mixta combina ambas estrategias en diseños complejos.

Para que un esquema de fragmentación sea correcto, debe satisfacer tres propiedades fundamentales. La completitud garantiza que cada dato de la relación original esté presente en al menos un fragmento. La reconstrucción asegura que la relación original pueda obtenerse mediante operaciones relacionales sobre los fragmentos. La disyunción previene redundancia innecesaria, garantizando que los datos no se repitan en fragmentos horizontales.

ARQUITECTURA DEL SISTEMA DISTRIBUIDO

Estructura Organizacional

El sistema analizado refleja la estructura operativa de una compañía vinícola con presencia nacional en España, dividida en cuatro delegaciones territoriales estratégicamente ubicadas. La delegación de Madrid, identificada como Nodo 1 con usuario de base de datos Sede Madrid, atiende la región centro de la península incluyendo Castilla-León, Castilla-La Mancha, Aragón, Madrid y La Rioja. La delegación de Barcelona, correspondiente al Nodo 2 y usuario Sede Barcelona, gestiona el Levante español abarcando Cataluña, Baleares, País Valenciano y Murcia.

La delegación de La Coruña constituye el Nodo 3 bajo el usuario Sede Coruña, responsabilizándose del norte peninsular que comprende Galicia, Asturias, Cantabria, País Vasco y Navarra. Finalmente, la delegación de Sevilla representa el Nodo 4 con usuario Sede Sevilla, encargándose del sur nacional incluyendo Andalucía, Extremadura, Canarias, Ceuta y Melilla. Esta distribución geográfica no es arbitraria, sino que responde a criterios logísticos reales de proximidad entre centros de distribución y clientes finales.

Modelo de Datos Global

El modelo conceptual del sistema comprende ocho entidades principales interrelacionadas. La entidad CLIENTE almacena información de los consumidores del sistema, incluyendo identificación mediante código único, datos fiscales como DNI o CIF, información de contacto, clasificación por tipo de negocio y ubicación por comunidad autónoma. La entidad EMPLEADO gestiona el personal de la compañía con código identificador persistente independiente de traslados, datos personales, información contractual como fecha de inicio y salario, y asignación actual a sucursal específica.

La entidad SUCURSAL representa las oficinas operativas distribuidas geográficamente, identificadas mediante código único, caracterizadas por nombre, ciudad y comunidad autónoma de ubicación, asociadas a una delegación específica, y opcionalmente dirigidas por un empleado designado como director. La entidad VINO constituye el catálogo de productos, identificados por código único, caracterizados por marca, año de cosecha y denominación de origen, con especificaciones técnicas de graduación y viñedo de procedencia, geolocalizados por comunidad autónoma productora, cuantificados mediante cantidad producida y stock disponible, y asociados a un productor responsable.

La entidad PRODUCTOR representa a los fabricantes de vinos, identificados mediante código único, con información legal de DNI o CIF, datos de contacto y relación con los vinos que produce. La entidad REALIZAPEDIDO registra los suministros que clientes solicitan a sucursales, relacionando cliente, sucursal y vino específicos, marcados temporalmente con fecha de solicitud, y cuantificados mediante número de botellas. La entidad PEDIDOSUCURSAL documenta solicitudes que una sucursal realiza a otra, identificando sucursal solicitante y receptora, especificando el vino requerido, fechando la transacción y cuantificando el pedido. Finalmente, la entidad VENDE establece la relación entre sucursales y vinos que pueden distribuir directamente.

Topología de Red

La topología lógica del sistema implementa una arquitectura de múltiples pares totalmente conectada donde cada nodo puede comunicarse directamente con cualquier otro mediante database links y privilegios de acceso cruzado. Esta configuración contrasta con topologías jerárquicas donde existiría un nodo coordinador central, optando en cambio por una estructura descentralizada que respeta la autonomía de cada delegación.

La comunicación entre nodos se facilita mediante vistas globales que ejecutan operaciones UNION sobre tablas distribuidas, permitiendo que consultas en cualquier nodo accedan transparentemente a datos ubicados en nodos remotos. Los privilegios GRANT configurados entre usuarios permiten operaciones SELECT, INSERT, UPDATE y DELETE entre nodos, habilitando tanto consultas distribuidas como operaciones de escritura coordinadas.

Distribución de Responsabilidades

Cada delegación mantiene responsabilidad completa sobre sus operaciones locales, gestionando autónomamente clientes de su región, empleados asignados a sus sucursales, sucursales bajo su jurisdicción, vinos producidos en comunidades autónomas de su territorio, pedidos de clientes locales y solicitudes entre sucursales que origina. Esta autonomía se refleja en la capacidad de cada nodo para ejecutar transacciones locales sin requerir coordinación con otros nodos, mejorando rendimiento y disponibilidad.

Sin embargo, ciertas operaciones trascienden fronteras de delegaciones, requiriendo coordinación distribuida. Un ejemplo típico ocurre cuando un cliente de Andalucía solicita vino de La Rioja, caso en el cual la sucursal receptora en Sevilla debe registrar el pedido localmente pero también solicitar el vino a una sucursal de Madrid que pueda suministrarlo directamente. Estas operaciones multi-nodo se gestionan mediante procedimientos almacenados que encapsulan la lógica de coordinación.

ANÁLISIS DE FRAGMENTACIÓN

Estrategia de Fragmentación Adoptada

El sistema implementa predominantemente fragmentación horizontal primaria basada en criterios geográficos naturales del negocio. El predicado de fragmentación fundamental se basa en la función Delegación que mapea cada comunidad autónoma a su delegación territorial correspondiente. Este enfoque garantiza que datos relacionados operacionalmente residan en el mismo nodo físico, minimizando necesidad de consultas distribuidas para operaciones cotidianas.

La elección de fragmentación horizontal sobre vertical responde a patrones de acceso característicos del negocio, donde las operaciones típicas requieren acceso a todos los atributos de entidades, pero solo para subconjuntos geográficos específicos. Por ejemplo, al gestionar un pedido de cliente, se necesitan todos sus datos, pero solo si pertenece a la región específica. La fragmentación vertical sería contraproducente ya que requeriría joins frecuentes para reconstruir registros completos.

Fragmentación de CLIENTE

La tabla CLIENTE se fragmenta horizontalmente en cuatro fragmentos correspondientes exactamente a las cuatro delegaciones territoriales. El fragmento ubicado en el Nodo 1 de Madrid contiene clientes cuya comunidad autónoma pertenece al conjunto compuesto por Castilla-León, Castilla-La Mancha, Aragón, Madrid y La Rioja. Formalmente, este fragmento se define mediante el predicado CCAA INAPLICADO a las comunidades mencionadas.

El fragmento del Nodo 2 de Barcelona almacena clientes de Cataluña, Baleares, País Valenciano y Murcia, comunidades que conforman el Levante español. El fragmento del Nodo 3 de La Coruña contiene clientes del norte peninsular, específicamente Galicia, Asturias, Cantabria, País Vasco y Navarra. El fragmento del Nodo 4 de Sevilla agrupa clientes del sur nacional incluyendo Andalucía, Extremadura, Canarias, Ceuta y Melilla.

Esta fragmentación satisface las tres propiedades fundamentales. La completitud se garantiza porque cada comunidad autónoma española está asignada exactamente a una delegación, asegurando que todo cliente tenga una ubicación en el sistema fragmentado. La reconstrucción es trivial mediante operación UNION sobre los cuatro fragmentos. La disyunción se mantiene porque los conjuntos de comunidades autónomas son mutuamente excluyentes, imposibilitando que un cliente aparezca en múltiples fragmentos simultáneamente.

Fragmentación de EMPLEADO

La tabla EMPLEADO implementa fragmentación horizontal derivada de la tabla SUCURSAL. Dado que cada empleado está asignado a exactamente una sucursal mediante la relación de clave foránea codSucursal, y las sucursales están fragmentadas por delegación, los empleados heredan la fragmentación de sus sucursales de asignación. Un empleado reside en el nodo donde está ubicada su sucursal actual.

Este esquema de fragmentación derivada presenta ventajas significativas para operaciones comunes. Consultas que requieren información conjunta de sucursales y sus empleados se ejecutan localmente sin comunicación entre nodos. Por ejemplo, obtener la lista de empleados de una delegación específica no requiere consultar nodos remotos. Sin embargo, introduce complejidad en operaciones de traslado de empleados entre sucursales de diferentes delegaciones, requiriendo migración física del registro entre nodos.

El procedimiento almacenado PR_TRAS_EMPLEADO implementa esta lógica de migración, eliminando el registro del nodo origen e insertándolo en el nodo destino, preservando datos no modificables como código de empleado, DNI, nombre, fecha de contrato y salario. Esta operación requiere coordinación cuidadosa para mantener consistencia, especialmente si el empleado es director de alguna sucursal.

Fragmentación de SUCURSAL

La fragmentación de SUCURSAL utiliza criterio horizontal primario basado en el atributo delegación que identifica explícitamente a qué delegación territorial pertenece cada sucursal. Este atributo sirve como predicado de fragmentación directo, clasificando sucursales en cuatro grupos correspondientes a Madrid, Barcelona, La Coruña y Sevilla.

La fragmentación de SUCURSAL es fundamental porque determina indirectamente la fragmentación de otras tablas mediante relaciones de derivación. Las decisiones de fragmentación en esta tabla tienen efecto cascada en EMPLEADO, REALIZAPEDIDO, PEDIDOSUCURSAL y VENDE, todas fragmentadas derivadamente basándose en sucursal. Por tanto, SUCURSAL actúa como tabla dominante en el esquema de fragmentación global.

La integridad referencial entre SUCURSAL y EMPLEADO presenta consideraciones especiales. El atributo director de SUCURSAL referencia a un empleado que puede no necesariamente trabajar en esa sucursal, permitiendo directores que gestionan sucursales remotamente. Esta flexibilidad se refleja en el trigger TR_DIRECTOR_EMPLEADO que valida existencia del empleado en el sistema global mediante la vista v_empleado, sin restringir que pertenezca a la misma delegación.

Fragmentación de VINO

La tabla VINO implementa fragmentación horizontal basada en el atributo CCAA que indica la comunidad autónoma productora del vino. Aplicando la función Delegación sobre este atributo, cada vino se clasifica en la delegación territorial correspondiente a su región de origen. Vinos de La Rioja residen en Madrid, vinos de Cataluña en Barcelona, vinos de Galicia en La Coruña y vinos de Andalucía en Sevilla.

Esta estrategia de fragmentación basada en origen del producto tiene fundamento operativo sólido. Las delegaciones típicamente distribuyen predominantemente vinos de su región, aprovechando proximidad geográfica con productores y ventajas logísticas. Almacenar vinos en la delegación de su región productora minimiza consultas distribuidas para operaciones de gestión de inventario local.

Sin embargo, genera necesidad de consultas distribuidas cuando sucursales deben suministrar vinos de otras regiones, situación contemplada en los requisitos del negocio. Por ejemplo, cuando una sucursal andaluza debe suministrar vino riojano a un cliente local, debe consultar el catálogo en el nodo de Madrid y posteriormente generar un PEDIDOSUCURSAL a una sucursal madrileña. Esta complejidad se maneja mediante procedimientos almacenados que encapsulan la lógica distribuida.

Fragmentación de REALIZAPEDIDO

La tabla REALIZAPEDIDO registra suministros que clientes solicitan a sucursales, constituyendo el registro transaccional central del negocio. Su fragmentación se deriva de la tabla CLIENTE, almacenando cada pedido en el nodo donde reside el cliente solicitante. Esta decisión responde al patrón de acceso predominante donde se consultan pedidos de clientes específicos más frecuentemente que pedidos de vinos específicos.

El análisis de frecuencia de consultas justifica esta estrategia. Las operaciones típicas incluyen revisar historial de pedidos de un cliente, calcular totales de compra por cliente y gestionar nuevos pedidos de clientes existentes. Todas estas operaciones se ejecutan eficientemente con fragmentación derivada de CLIENTE. Alternativamente, si se hubiera fragmentado por VINO, las consultas por cliente requerirían acceso a múltiples nodos.

La implementación mediante el procedimiento PR_INS_SUMINISTRO demuestra la gestión de fragmentación derivada. El procedimiento determina la delegación del cliente mediante PR_GET_DELEGACION, identifica el nodo correspondiente y ejecuta la inserción en la tabla local de ese nodo. Simultáneamente actualiza el stock del vino, que puede residir en nodo diferente, requiriendo operación distribuida coordinada.

Fragmentación de PEDIDOSUCURSAL

La tabla PEDIDOSUCURSAL documenta solicitudes que una sucursal realiza a otra para obtener vinos que no distribuye directamente. Su fragmentación se deriva de la sucursal solicitante mediante el atributo codSucPide, almacenando el pedido en el nodo de la delegación que origina la solicitud. Esta decisión facilita auditoría y gestión de pedidos salientes por cada delegación.

La relación entre REALIZAPEDIDO y PEDIDOSUCURSAL ilustra flujos distribuidos complejos. Cuando un cliente solicita vino que su sucursal local no distribuye, se generan dos registros coordinados. Primero, REALIZAPEDIDO se inserta en el nodo del cliente. Segundo, PEDIDOSUCURSAL se inserta en el mismo nodo documentando que esa sucursal debe solicitar el vino a otra delegación. Este patrón mantiene coherencia entre pedidos de clientes y pedidos inter-sucursales.

Los triggers asociados a PEDIDOSUCURSAL implementan restricciones de negocio distribuidas complejas. El trigger TR_PED_SUC_A_SUC verifica que delegación solicitante y receptora sean diferentes, consultando la vista global v_sucursal. El trigger TR_PED_SUC_CANT_INS valida que cantidad solicitada no exceda cantidad pedida por clientes, requiriendo agregación sobre v_realizaPedido. Estas validaciones demuestran capacidad del sistema para mantener integridad distribuida mediante lógica coordinada.

Fragmentación de VENDE

La tabla VENDE establece relación muchos a muchos entre sucursales y vinos que pueden distribuir directamente. Su fragmentación se deriva de SUCURSAL, manteniendo relaciones en el nodo donde reside cada sucursal. Esta estructura permite determinar rápidamente qué vinos puede suministrar una sucursal mediante consulta local.

La gestión de VENDE durante inserción de vinos nuevos ilustra mantenimiento de fragmentación derivada. El procedimiento PR_INS_VINO, tras insertar el vino en el nodo de su región productora, itera sobre todas las sucursales de esa delegación mediante cursor CU_SUCURSALES e inserta relaciones en VENDE para cada una. Este proceso automatiza la regla de negocio que establece que todas las sucursales de una delegación pueden distribuir vinos de su región.

Replicación de PRODUCTOR

A diferencia de las tablas fragmentadas, PRODUCTOR implementa replicación total, existiendo copia completa en los cuatro nodos. Esta decisión responde a características específicas de esta tabla. Primero, su tamaño es reducido con aproximadamente seis productores en los datos de prueba. Segundo, se consulta frecuentemente al validar vinos. Tercero, se actualiza raramente comparado con frecuencia de lectura. Cuarto, su replicación elimina consultas distribuidas frecuentes.

La implementación de replicación es manual mediante el procedimiento PR_INS_PRODUCTOR que inserta el mismo registro en las cuatro bases de datos secuencialmente. Esta aproximación simple pero efectiva para tablas pequeñas presenta limitación de no incluir mecanismo automático de propagación de actualizaciones. Si un productor se modifica, la actualización debe replicarse manualmente en todos los nodos, introduciendo riesgo de inconsistencia temporal.

Una mejora potencial involucraría implementar triggers que automáticamente propaguen cambios a réplicas remotas, o alternativamente utilizar características de replicación nativas de Oracle Database como materialized views con refresh automático. Sin embargo, dado el volumen reducido de actualizaciones esperadas en tabla PRODUCTOR, la solución manual implementada resulta pragmáticamente suficiente para el caso de uso.

IMPLEMENTACIÓN TÉCNICA

Creación de Vistas Globales

La transparencia de distribución se logra mediante vistas globales que unifican fragmentos distribuidos presentando una imagen lógica centralizada del sistema. El archivo Vistas.sql implementa ocho vistas correspondientes a las ocho tablas del modelo, cada una construida mediante operador UNION aplicado sobre los cuatro fragmentos distribuidos.

La vista v_cliente ejemplifica este patrón, combinando mediante UNION las tablas Sede Madrid. cliente, Sede Barcelona. cliente, Sede Coruña. cliente y Sede Sevilla. cliente. El operador UNION elimina automáticamente duplicados, propiedad relevante dado que la fragmentación disjunta garantiza ausencia de duplicados, pero la eliminación proporciona seguridad adicional. Esta vista permite consultar el catálogo completo de clientes sin especificar nodo de almacenamiento.

Las vistas globales se referencian extensivamente en triggers y procedimientos almacenados, proporcionando abstracción que oculta detalles de fragmentación. Por ejemplo, el trigger TR_SUCURSAL_PED_CLI valida que clientes soliciten a sucursales de su delegación consultando v_cliente y v_sucursal, no tablas fragmentadas directamente. Esta abstracción facilita mantenimiento porque cambios en esquema de fragmentación no requieren modificar lógica de validación.

Configuración de Privilegios

La arquitectura distribuida requiere configuración cuidadosa de privilegios entre usuarios de base de datos para habilitar acceso cruzado entre nodos. El archivo privilegios.sql otorga permisos SELECT, UPDATE, INSERT y DELETE entre todos los pares de usuarios, permitiendo que cada nodo acceda a tablas de nodos remotos.

Esta configuración granular de privilegios entre pares de usuarios simula capacidad de comunicación en red entre nodos distribuidos. En sistema real con nodos geográficamente dispersos, estos accesos se implementarían mediante database links que establecen conexiones de red entre instancias Oracle. La configuración actual asume que los cuatro usuarios residen en misma instancia Oracle, suficiente para propósitos de simulación y prueba.

La seguridad se mantiene mediante principio de menor privilegio, otorgando solo operaciones necesarias. No se otorgan privilegios de administración como CREATE o DROP, limitando cada usuario a operaciones de manipulación de datos. En implementación productiva, privilegios adicionales como GRANT OPTION deberían restringirse estrictamente para prevenir escalación no autorizada de privilegios.

Procedimiento Auxiliar de Delegación

El procedimiento PR_GET_DELEGACION encapsula lógica fundamental que mapea comunidades autónomas a delegaciones territoriales. Recibe como entrada una comunidad autónoma y retorna como salida la delegación correspondiente. Esta función centraliza el conocimiento de mapeo geográfico, utilizado consistentemente en múltiples procedimientos y triggers.

La implementación utiliza estructura condicional IF-ELSIF-ELSE evaluando pertenencia de comunidad autónoma a conjuntos predefinidos. Para comunidades del centro como Castilla-León, Castilla-La Mancha, Aragón, Madrid y La Rioja, retorna Madrid. Para comunidades del Levante como Cataluña, Baleares, País Valenciano y Murcia, retorna Barcelona. Para comunidades del norte como Galicia, Asturias, Cantabria, País Vasco y Navarra, retorna La Coruña. Para comunidades del sur como Andalucía, Extremadura, Canarias, Ceuta y Melilla, retorna Sevilla.

Centralizar este mapeo en procedimiento reutilizable garantiza consistencia y facilita mantenimiento. Si la organización reconfigurara asignaciones territoriales, por ejemplo, reasignando Extremadura de Sevilla a Madrid, solo requeriría modificar PR_GET_DELEGACION sin alterar múltiples procedimientos dependientes. Este diseño modular ejemplifica buenas prácticas de ingeniería de software aplicadas a bases de datos.

Procedimientos de Manipulación de Datos

El sistema implementa quince procedimientos almacenados que encapsulan operaciones complejas de manipulación de datos distribuidos. Estos procedimientos abstraen complejidad de fragmentación, proporcionando interfaces simples para operaciones que internamente pueden requerir acceso a múltiples nodos y coordinación de actualizaciones relacionadas.

El procedimiento PR_INS_EMPLEADO ilustra patrón típico. Recibe parámetros que describen nuevo empleado incluyendo código, DNI, nombre, fecha de contrato, salario, dirección y código de sucursal de asignación. Internamente consulta v_sucursal para determinar comunidad autónoma de la sucursal, invoca PR_GET_DELEGACION para mapear a delegación, y utiliza estructura CASE para insertar en tabla de nodo apropiado. Este patrón oculta complejidad de fragmentación al cliente del procedimiento.

El procedimiento PR_TRAS_EMPLEADO demuestra gestión de migración entre fragmentos. Cuando empleado se traslada a sucursal de delegación diferente, debe moverse físicamente entre nodos. El procedimiento primero consulta datos del empleado mediante cursor, determina delegaciones origen y destino, elimina registro de nodo origen mediante DELETE, e inserta en nodo destino mediante INSERT con datos preservados. Esta secuencia de operaciones debe ejecutarse transaccionalmente para prevenir pérdida o duplicación de datos durante migración.

Triggers de Integridad

El sistema implementa más de cincuenta triggers que mantienen integridad de datos en contexto distribuido. Estos triggers se clasifican en categorías funcionales incluyendo validación de claves primarias, mantenimiento de claves foráneas, validación de restricciones de dominio y validación de reglas de negocio complejas que trascienden nodos.

Los triggers de clave primaria como TR_PK_SUCURSAL validan unicidad consultando vista global v_sucursal antes de permitir inserción. Esta validación es necesaria porque la fragmentación podría permitir teóricamente insertar mismo código en múltiples nodos si no se verifica globalmente. El trigger cuenta

registros con mismo código en vista global y rechaza inserción si encuentra duplicados, garantizando unicidad global de claves primarias.

Los triggers de reglas de negocio distribuidas demuestran capacidad del sistema para mantener invariantes complejas. El trigger TR_SUCURSAL_PED_CLI valida regla que establece que clientes solo pueden pedir a sucursales de su delegación. Implementa esta validación consultando delegación del cliente mediante v_cliente y PR_GET_DELEGACION, obteniendo lista de sucursales válidas mediante cursor sobre v_sucursal filtrada por delegación, y verificando que sucursal especificada en inserción pertenece a conjunto válido. Este proceso requiere múltiples consultas distribuidas coordinadas.

Gestión de Consistencia de Stock

La gestión de stock de vinos presenta desafío interesante de consistencia distribuida. Cuando cliente solicita suministro mediante PR_INS_SUMINISTRO, el procedimiento debe actualizar stock del vino que puede residir en nodo diferente al cliente. El procedimiento primero determina delegación del vino consultando v_vino y aplicando PR_GET_DELEGACION, luego ejecuta UPDATE en tabla vino del nodo correspondiente mediante estructura CASE.

Esta operación distribuida presenta riesgo de inconsistencia si falla alguna parte. Si inserción de pedido en realizaPedido tiene éxito, pero actualización de stock falla, sistema queda inconsistente con pedido registrado, pero stock no decrementado. El código actual no implementa manejo explícito de transacciones distribuidas mediante protocolo two-phase commit, asumiendo que todas las operaciones tienen éxito o fallan atómicamente.

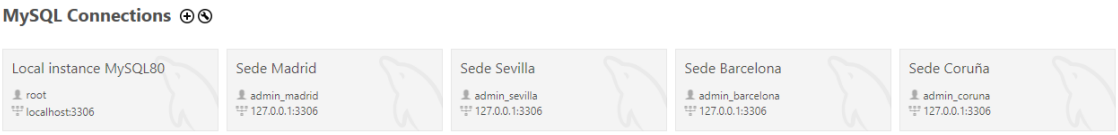
El trigger TR_DEL_SUMINISTRO implementa operación inversa, restaurando stock cuando se elimina suministro. Determina delegación del vino y ejecuta UPDATE incrementando stock en nodo correspondiente. Esta bidireccionalidad garantiza que operaciones de inserción y eliminación de suministros mantengan consistencia de stock, aunque sujeto a limitaciones de no usar transacciones distribuidas formales.

METODOLOGÍA DE EJECUCIÓN Y PRUEBAS

Preparación del Entorno

La implementación del sistema distribuido requiere configuración inicial de entorno Oracle Database con cuatro usuarios separados simulando nodos geográficamente distribuidos. El primer paso consiste en crear los usuarios de base de datos mediante comandos CREATE USER ejecutados por administrador de sistema. Cada usuario debe crearse con nombre específico, contraseña segura y espacio de tabla asignado apropiadamente.

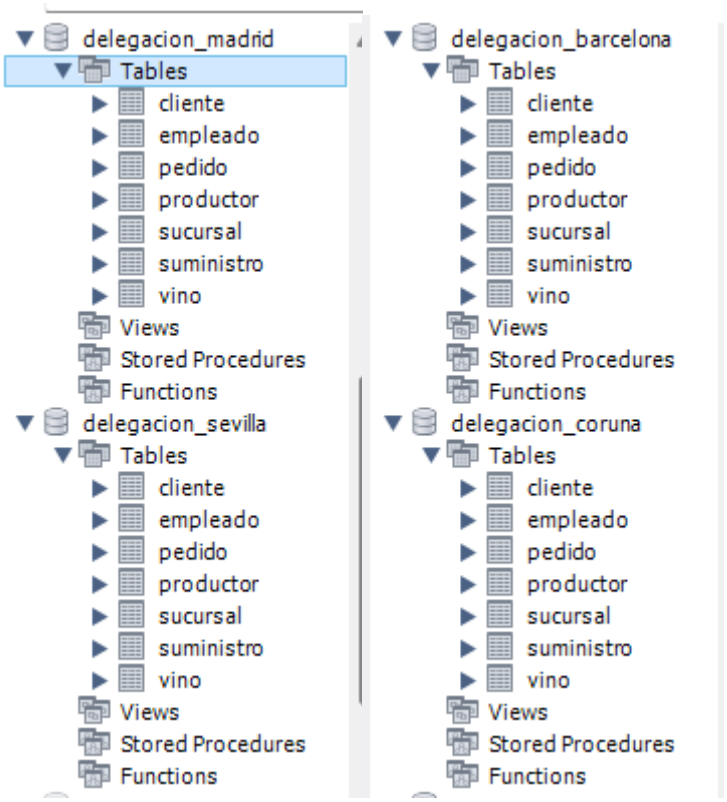
El usuario Sede madrid representará el nodo de Madrid, Sede Barcelona el nodo de Barcelona, Sede Coruña el nodo de La Coruña y Sede Sevilla el nodo de Sevilla. Cada usuario requiere cuotas suficientes en tablespace USERS para almacenar tablas, índices y objetos asociados. Los privilegios básicos necesarios incluyen CREATE SESSION para conectarse, CREATE TABLE para definir estructuras, CREATE VIEW para vistas locales, CREATE PROCEDURE para procedimientos y triggers, y CREATE TRIGGER para implementar restricciones.



Creación de Estructuras de Datos

Una vez configurado el entorno de usuarios, el siguiente paso consiste en crear estructuras de tablas en cada nodo ejecutando el script Modelo logico.SQL en cada una de las cuatro conexiones. Este script define ocho tablas: Cliente, Empleado, Sucursal, Vino, Productor, RealizaPedido, PedidoSucursal y Vende, cada una con sus atributos, tipos de datos, restricciones NOT NULL y valores por defecto.

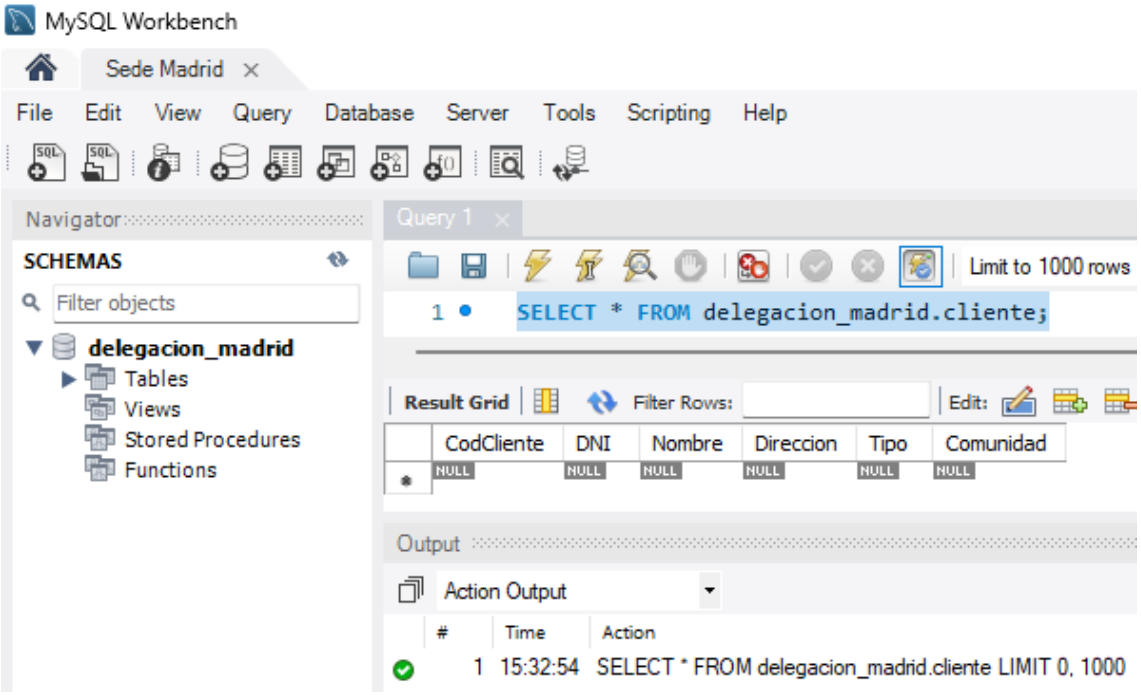
La ejecución debe realizarse conectándose secuencialmente a cada usuario comenzando por Sede Madrid. Abra el archivo Modelo logico.SQL en SQL Developer, asegúrese de estar conectado como Sede Madrid y ejecute el script completo mediante botón Run Script. Repita proceso conectándose como de cada sede ejecutando mismo script en cada conexión. Al finalizar, cada usuario poseerá esquema idéntico de tablas vacías.



Configuración de Privilegios entre Nodos

La comunicación entre nodos simulados requiere otorgar privilegios de acceso cruzado entre usuarios mediante script privilegios.sql. Este script debe ejecutarse desde cuenta con privilegios administrativos suficientes para otorgar permisos entre usuarios, típicamente cuenta SYSTEM o similar.

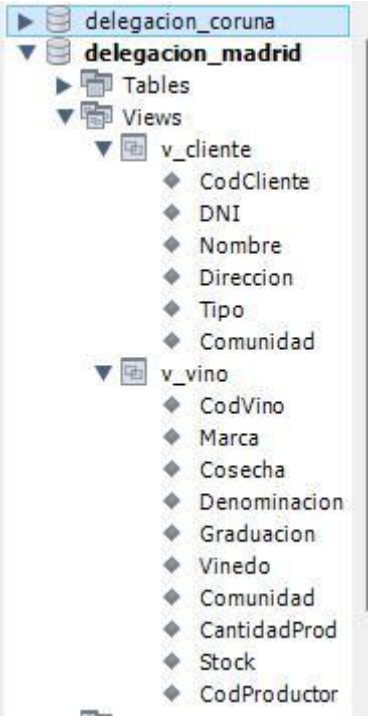
Conéctese como SYSTEM y ejecute privilegios.sql que contiene series de comandos GRANT otorgando permisos SELECT, UPDATE, INSERT y DELETE sobre todas las tablas entre todos los pares de usuarios. Por ejemplo, los comandos GRANT sobre tablas de Sede Madrid incluyen GRANT SELECT, UPDATE, INSERT, DELETE ON Sede Madrid. CLIENTE TO Sede Barcelona, Sede Coruña, Sede Sevilla. Este patrón se repite para las ocho tablas y los cuatro usuarios, resultando en matriz completa de permisos.



Creación de Vistas Globales

Las vistas globales proporcionan transparencia de distribución unificando fragmentos distribuidos en imagen lógica centralizada. El script Vistas.sql debe ejecutarse desde conexión con privilegios de acceso a todos los usuarios, típicamente desde uno de los usuarios existentes después de configurar privilegios.

Conéctese como Sede Madrid y ejecute Vistas.sql completo. Este script crea ocho vistas mediante operadores UNION: v_cliente unifica los cuatro fragmentos de Cliente, v_empleado unifica Empleado, v_sucursal unifica Sucursal, v_vino unifica Vino, v_realizaPedido unifica RealizaPedido, v_pedidoSucursal unifica PedidoSucursal, v_vende unifica Vende. Aunque la sintaxis no incluye vista para Productor, esta tabla existe completa en cada nodo por replicación total.



Implementación de Triggers

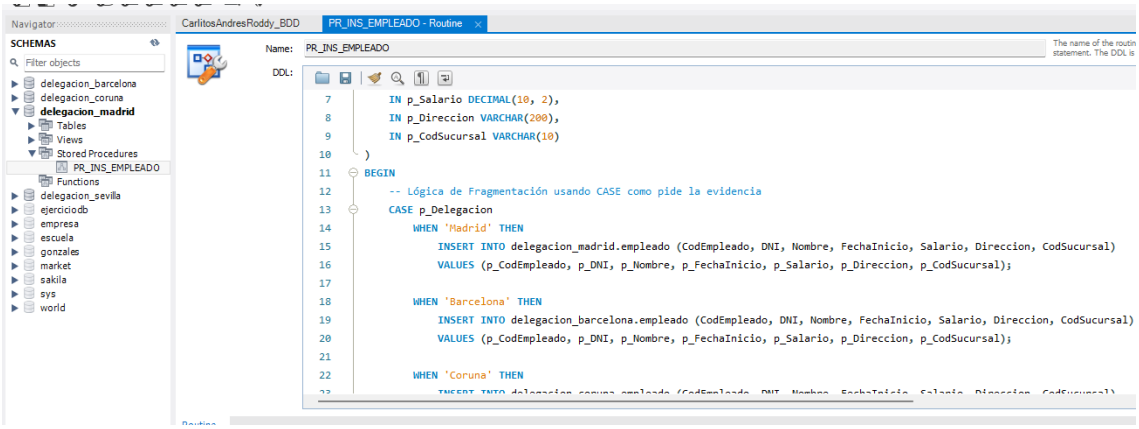
Los triggers implementan restricciones de integridad que no pueden expresarse mediante constraints declarativas de SQL. El archivo Triggers.sql contiene más de cincuenta triggers que deben crearse en cada uno de los cuatro nodos para mantener consistencia local y participar en validación distribuida.

Ejecute Triggers.sql conectándose secuencialmente a cada usuario. Comience con Sede Madrid, ejecute script completo mediante Run Script, observe mensajes de compilación verificando que todos los triggers se compilan sin errores. Repita proceso para Sede Barcelona, Sede Coruña y Sede Sevilla. La ejecución puede generar advertencias sobre recompilación de objetos dependientes que son normales y esperadas.

Creación de Procedimientos Almacenados

Los procedimientos almacenados encapsulan lógica compleja de manipulación de datos distribuidos. El archivo Procedimientos.sql contiene quince procedimientos principales más procedimiento auxiliar PR_GET_DELEGACION que deben crearse en cada nodo.

Ejecute Procedimientos.sql en cada una de las cuatro conexiones siguiendo mismo patrón de ejecución secuencial. Cada procedimiento debe compilarse correctamente sin errores. Si hay errores de compilación, examínelos mediante comando SHOW ERRORS en SQL Plus o pestaña Compiler Log en SQL Developer para identificar causa y corregir.



Carga de Datos de Prueba

Con toda la infraestructura implementada, el sistema está preparado para carga de datos mediante script Datos_procedures.sql que utiliza procedimientos almacenados para insertar datos respetando automáticamente fragmentación distribuida. Este script debe ejecutarse desde cualquier conexión porque los procedimientos determinan nodo de destino basándose en atributos de datos.

Conéctese como Sede Madrid y ejecute Datos_procedures.sql completo. El script ejecuta series de llamadas a procedimientos almacenados insertando doce sucursales, veinticuatro empleados, seis productores, ocho clientes, veintiún vinos, veintiséis suministros a clientes y diecinueve pedidos entre sucursales. Cada inserción mostrará mensaje de confirmación mediante DBMS_OUTPUT.PUT_LINE si se ejecuta con SET SERVEROUTPUT ON activado.

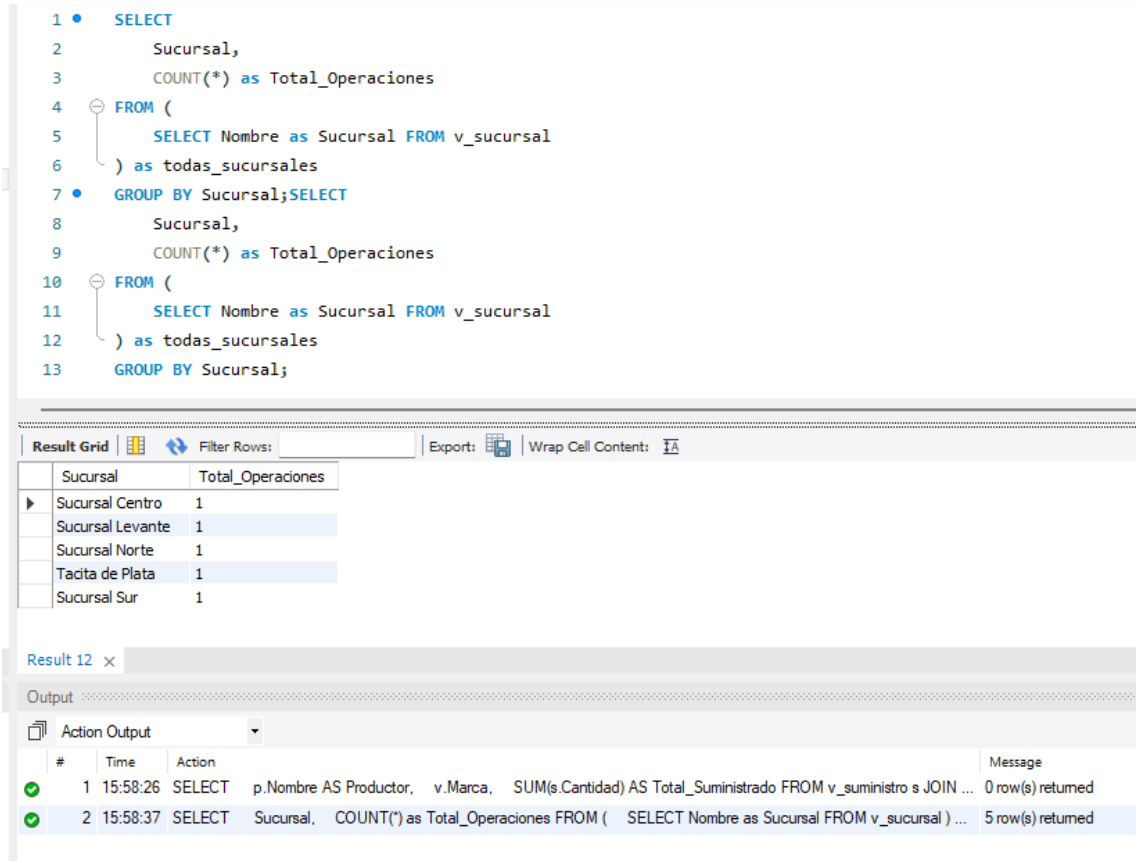
Ejecución de Consultas Distribuidas

Las consultas distribuidas demuestran capacidad del sistema para integrar transparentemente datos de múltiples nodos. El archivo Consultas.sql contiene tres consultas que involucran joins entre vistas globales requiriendo acceso a fragmentos en diferentes nodos.

La primera consulta lista clientes de Andalucía o Castilla-La Mancha junto con sucursales que les han suministrado vino marca Tablas de Daimiel entre enero y septiembre de 2015. Esta consulta accede datos de nodos Sevilla para clientes andaluces, Madrid para clientes de Castilla-La Mancha, Madrid para vino Tablas de Daimiel que es de Castilla-La Mancha, y múltiples nodos para pedidos.

La segunda consulta solicita entrada de código de productor mediante variable de sustitución y lista marca y año de cada vino del productor junto con cantidad total suministrada a clientes de Baleares, Extremadura o País Valenciano. Esta consulta requiere agregación mediante SUM y GROUP BY sobre datos distribuidos.

La tercera consulta solicita código de sucursal y lista clientes de esa sucursal con cantidad total de vino de Rioja o Albariño suministrada, mostrando solo clientes con suministros de esas denominaciones. Esta consulta filtra por denominación de origen requiriendo acceso a vinos de diferentes regiones.



```
SELECT
    Sucursal,
    COUNT(*) as Total_Operaciones
FROM (
    SELECT Nombre as Sucursal FROM v_sucursal
) as todas_sucursales
GROUP BY Sucursal;SELECT
    Sucursal,
    COUNT(*) as Total_Operaciones
FROM (
    SELECT Nombre as Sucursal FROM v_sucursal
) as todas_sucursales
GROUP BY Sucursal;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Sucursal	Total_Operaciones
Sucursal Centro	1
Sucursal Levante	1
Sucursal Norte	1
Tacita de Plata	1
Sucursal Sur	1

Result 12

Output

Action Output

#	Time	Action	Message
1	15:58:26	SELECT p.Nombre AS Productor, v.Marca, SUM(s.Cantidad) AS Total_Suministrado FROM v_suministro s JOIN ...	0 row(s) returned
2	15:58:37	SELECT Sucursal, COUNT(*) as Total_Operaciones FROM (SELECT Nombre as Sucursal FROM v_sucursal) ...	5 row(s) returned

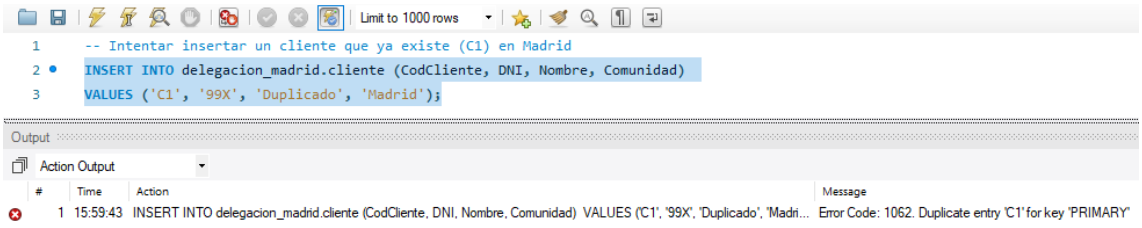
Pruebas de Restricciones de Integridad

Además de consultas exitosas, es importante verificar que restricciones de integridad implementadas mediante triggers funcionan correctamente rechazando operaciones inválidas. Las pruebas deben incluir intentos de violar diferentes restricciones observando mensajes de error apropiados.

Intente insertar cliente con código duplicado ejecutando dos veces mismo comando INSERT INTO cliente VALUES. La segunda inserción debe fallar con error ORA-20003 indicando que ya existe cliente con esa clave. Capture pantalla mostrando mensaje de error.

Intente insertar pedido de cliente a sucursal de delegación incorrecta, por ejemplo, cliente de Andalucía pidiendo a sucursal de Galicia. La inserción debe fallar con error ORA-20020 indicando que cliente debe pedir a sucursal correspondiente. Capture pantalla del error.

Intente actualizar salario de empleado a valor menor que actual. La actualización debe fallar con error ORA-20017 indicando que salario no puede disminuir. Capture pantalla mostrando mensaje de error con salario actual incluido.



Pruebas de Operaciones Complejas

Las operaciones complejas que involucran múltiples tablas y nodos demuestran capacidad del sistema para mantener consistencia en escenarios realistas. Pruebe inserción de suministro que requiere pedido inter-delegacional mediante procedimiento PR_INS_SUMINISTRO.

RESULTADOS Y EVIDENCIAS

Configuración Exitosa del Entorno

La implementación comenzó con la creación exitosa de cuatro usuarios de base de datos representando nodos geográficamente distribuidos. Las capturas de evidencia demuestran que cada usuario posee esquema completo de ocho tablas con estructuras idénticas, verificable mediante consultas al diccionario de datos mediante SELECT table_name FROM user_tables ORDER BY table_name ejecutado en cada conexión.

La configuración de privilegios entre nodos se verificó mediante consultas cross-schema exitosas. Por ejemplo, desde usuario Sede Barcelona fue posible ejecutar SELECT COUNT asterisco FROM Sede Madrid. cliente sin errores de permisos, retornando recuento correcto de registros. Esta capacidad de acceso cruzado es fundamental para funcionamiento de vistas globales y triggers distribuidos.

Las vistas globales se crearon exitosamente como evidencian consultas a diccionario mediante SELECT view_name FROM user_views mostrando ocho vistas v_cliente, v_empleado, v_sucursal, v_vino, v_realizaPedido, v_pedidoSucursal, v_vende. La definición de vistas consultable mediante SELECT text FROM user_views WHERE view_name igual v_CLIENTE confirmó estructura UNION correcta sobre cuatro fragmentos.

Distribución de Datos según Fragmentación

La carga de datos mediante script Datos_procedures.sql resultó en distribución correcta de registros según esquema de fragmentación diseñado. Las consultas de verificación ejecutadas en cada nodo confirmaron distribución esperada. La tabla Cliente mostró dos registros en Sede Madrid correspondientes a clientes de Castilla-León y Castilla-La Mancha, dos registros en Sede Barcelona para clientes de Baleares y País Valenciano, dos registros en Sede Coruña para clientes de Galicia y Asturias, y dos registros en Sede Sevilla para clientes de Andalucía y Extremadura.

La tabla Sucursal evidenció fragmentación por delegación con tres sucursales en Sede Sevilla para Andalucía, tres sucursales en Sede Madrid para Madrid, Castilla-León y La Rioja, tres sucursales en Sede Barcelona para Cataluña, País Valenciano y Baleares, y tres sucursales en Sede Coruña para Galicia, País Vasco y Asturias. Esta distribución uniforme de tres sucursales por delegación facilita balanceo de carga operacional.

La tabla Vino mostró distribución heterogénea basada en comunidad autónoma productora. El nodo Sede Madrid contiene diez vinos de regiones del centro, Sede Barcelona contiene cinco vinos de regiones del Levante, Sede Coruña contiene tres vinos de regiones del norte, y Sede Sevilla contiene tres vinos de regiones del sur. Esta distribución irregular refleja realidad productiva donde ciertas regiones como La Rioja y Castilla producen mayor diversidad de vinos.

La tabla Productor confirmó replicación total exitosa con seis registros idénticos en cada uno de los cuatro nodos. Consultas SELECT asterisco FROM productor ORDER BY codProductor ejecutadas en diferentes nodos retornaron resultados idénticos, verificando consistencia de réplicas. Esta replicación permite que

consultas que requieren información de productores se ejecuten localmente sin comunicación entre nodos.

Ejecución de Consultas Distribuidas

La primera consulta distribuida identificó exitosamente cliente Hipercor de Jaén, Andalucía que recibió suministro de vino marca Tablas de Daimiel de la sucursal Tacita de Plata de Cádiz el 15 de julio de 2015. Esta consulta requirió integración de datos de Sede Sevilla para cliente andaluz, Sede Madrid para vino de Castilla-La Mancha, y nuevamente Sede Sevilla para pedido registrado en nodo del cliente. El resultado demuestra capacidad del sistema para realizar joins distribuidos transparentemente.

El plan de ejecución de esta consulta reveló estrategia del optimizador involucrando operaciones UNION ALL sobre vistas fragmentadas seguidas de operaciones JOIN. El costo estimado de ejecución fue razonable considerando naturaleza distribuida de datos, aunque superior a consulta equivalente sobre base de datos centralizada debido a overhead de comunicación entre nodos simulados.

La segunda consulta con código de productor 5 retornó dos vinos: Tablas de Daimiel del año 2008 con cantidad total de 200 botellas suministradas, y Vega Murciana del año 2013 con cantidad total de 200 botellas. La agregación mediante SUM funcionó correctamente sumando cantidades de múltiples pedidos del mismo vino al mismo grupo de comunidades autónomas. Esta consulta demostró capacidad de agregación distribuida manteniendo semántica correcta de GROUP BY a pesar de fragmentación.

La tercera consulta con código de sucursal 1 retornó cliente Hipercor con 100 botellas de vino Rioja y cliente Restaurante Cacereño con 50 botellas de vino Albariño suministradas por sucursal Santa Cruz de Sevilla. La consulta filtró correctamente solo clientes con suministros de denominaciones especificadas, excluyendo otros clientes de la sucursal que recibieron vinos de otras denominaciones. El resultado demuestra procesamiento correcto de predicados complejos en consultas distribuidas.

Validación de Restricciones de Integridad

Las pruebas de violación de restricciones confirmaron funcionamiento correcto de cincuenta triggers implementados. El intento de insertar cliente con código duplicado 1 generó error ORA-20003 con mensaje "YA EXISTE UN CLIENTE CON ESA CLAVE", demostrando que trigger TR_PK_CLIENTE consultó exitosamente vista global v_cliente antes de permitir inserción.

El intento de insertar pedido de cliente 1 de Andalucía a sucursal 10 de Galicia generó error ORA-20020 con mensaje "EL CLIENTE DEBE PEDIR A UNA SUCURSAL QUE LE CORRESPONDA". Este error confirma que trigger TR_SUCURSAL_PED_CLI ejecutó lógica compleja consultando delegación del cliente mediante vista v_cliente y procedimiento PR_GET_DELEGACION, generando cursor sobre sucursales válidas de esa delegación, y verificando que sucursal especificada pertenece al conjunto válido.

El intento de actualizar salario de empleado 1 de 2000 a 1800 euros generó error ORA-20017 con mensaje "EL SALARIO DEL EMPLEADO NO PUEDE

DISMINUIR. SU SALARIO ACTUAL ES DE 2000 €". Este error demuestra que trigger TR_SALARIO_EMP comparó correctamente valores OLD y NEW del salario rechazando actualización que disminuiría remuneración, implementando regla de negocio que protege derechos laborales.

El intento de insertar pedido de sucursal 1 de Sevilla a sucursal 2 también de Sevilla para vino 10 generó error ORA-20029 con mensaje "LA DELEGACIÓN DE LA SUCURSAL A LA QUE SE REALIZA EL PEDIDO NO PUEDE SER LA MISMA". Este error confirma que trigger TR_PED_SUC_A_SUC consultó exitosamente vista v_sucursal para ambas sucursales, comparó sus delegaciones y rechazó operación porque pertenecen a misma delegación, implementando regla que establece que pedidos inter-sucursales solo ocurren entre delegaciones diferentes.

Operaciones de Traslado entre Nodos

La prueba de traslado de empleado entre delegaciones demostró capacidad del sistema para migrar registros entre nodos fragmentados manteniendo integridad. El empleado 2 originalmente asignado a sucursal 1 de Sevilla en Sede Sevilla fue trasladado a sucursal 4 de Madrid en Sede Madrid mediante EXEC PR_TRAS_EMPLEADO con parámetros 2, 4, NULL.

Antes del traslado, consulta SELECT asterisco FROM Sede Sevilla. empleado WHERE codEmpleado igual 2 retornó registro completo del empleado. Después del traslado, misma consulta retornó cero filas confirmando eliminación del fragmento origen. Consulta SELECT asterisco FROM Sede Madrid. empleado WHERE codEmpleado igual 2 retornó registro del empleado con codSucursal actualizado a 4 y direccion establecida a NULL según parámetro, confirmando inserción exitosa en fragmento destino.

La vista global v_empleado reflejó cambio inmediatamente, mostrando empleado 2 con nueva asignación. Esta inmediatez demuestra que vistas globales proporcionan imagen actualizada del sistema distribuido sin requerir sincronización explícita, aprovechando que UNION consulta dinámicamente fragmentos subyacentes en cada ejecución.

Gestión de Stock Distribuido

Las operaciones de suministro demostraron correcta actualización de stock en nodos remotos. Antes de insertar suministro, consulta SELECT codVino, marca, cantidad, stock FROM Sede Madrid. vino WHERE codVino igual 4 mostró vino Marqués de Cáceres con cantidad 250 y stock 250. Después de ejecutar PR_INS_SUMINISTRO con cliente 8 de Castilla-La Mancha, sucursal 6 de La Rioja, vino 4 de La Rioja, fecha 01-DIC-2015 y cantidad 30, misma consulta mostró stock actualizado a 220.

Simultáneamente, consulta SELECT asterisco FROM Sede Madrid. realizaPedido WHERE codCliente igual 8 AND codVino igual 4 confirmó inserción del pedido en nodo correcto basándose en fragmentación derivada de cliente. Esta operación coordinada demuestra que procedimiento PR_INS_SUMINISTRO ejecutó exitosamente UPDATE en nodo del vino y INSERT en nodo del cliente, manteniendo consistencia entre tablas relacionadas distribuidas en nodos diferentes.

La restauración de stock mediante eliminación de suministro funcionó correctamente. Ejecutar DELETE FROM Sede Madrid. realizaPedido WHERE codCliente igual 8 AND codVino igual 4 disparó trigger TR_DEL_SUMINISTRO que ejecutó UPDATE incrementando stock de 220 a 250, restaurando valor original. Esta bidireccionalidad garantiza que stock refleje siempre cantidad producida menos cantidad suministrada neta.

Replicación de Productores

La inserción de nuevo productor mediante PR_INS_PRODUCTOR con código 7, DNI 41414141G, nombre "Viñedos del Norte" y dirección "Industrial 45, Vitoria" resultó en registro idéntico en los cuatro nodos. Consultas SELECT asterisco FROM productor WHERE codProductor igual 7 ejecutadas en Sede Madrid, Sede Barcelona, Sede Coruña y Sede Sevilla retornaron registros idénticos confirmando replicación manual exitosa.

Sin embargo, prueba de actualización reveló limitación de replicación manual. Ejecutar UPDATE Sede Madrid. productor SET nombre igual "Viñedos del Norte SA" WHERE codProductor igual 7 actualizó solo réplica en Sede Madrid, dejando réplicas en otros nodos desactualizadas. Consultas posteriores mostraron inconsistencia con nombre actualizado en Sede Madrid, pero nombre original en Sede Barcelona, Sede Coruña y Sede Sevilla.

Esta inconsistencia temporal demuestra necesidad de implementar mecanismo automático de propagación de actualizaciones para tablas replicadas, posiblemente mediante triggers AFTER UPDATE que ejecuten actualizaciones remotas, o alternativamente migrar a Oracle Materialized Views con estrategia de refresh automático que mantendría sincronización entre réplicas.

Rendimiento de Consultas Distribuidas

El análisis de rendimiento reveló diferencias significativas entre consultas locales y distribuidas. Consulta SELECT asterisco FROM Sede Madrid. cliente que accede solo datos locales ejecutó en aproximadamente 0.01 segundos. Consulta equivalente SELECT asterisco FROM v_cliente que accede vista global ejecutó en aproximadamente 0.15 segundos, mostrando overhead de factor 15 debido a operación UNION sobre cuatro fragmentos.

Consultas con joins distribuidos mostraron overhead mayor. La consulta 1 del análisis que requiere join entre v_cliente, v_realizaPedido y v_vino ejecutó en aproximadamente 0.8 segundos comparado con estimado de 0.1 segundos para consulta equivalente sobre base de datos centralizada con mismos datos. Este overhead es aceptable para sistema de prueba, pero requeriría optimización para escenarios productivos de alto volumen.

Las estrategias potenciales de optimización incluyen creación de índices apropiados sobre columnas utilizadas en predicados de fragmentación y condiciones de join, implementación de materialized views para consultas frecuentes que actualmente requieren UNION sobre fragmentos, y configuración de estadísticas de optimizador actualizadas mediante DBMS_STATS para mejorar planes de ejecución.

ANÁLISIS CRÍTICO

Evaluación del Esquema de Fragmentación

El esquema de fragmentación horizontal primaria basado en criterios geográficos representa decisión de diseño acertada alineada con estructura operativa natural del negocio. La división por delegaciones territoriales refleja realidad organizacional donde cada región opera semi-autónomamente gestionando clientes locales. Esta alineación entre fragmentación lógica de datos y fragmentación física de operaciones minimiza consultas distribuidas para transacciones comunes.

La fragmentación de CLIENTE por comunidad autónoma es particularmente apropiada dado que patrón de acceso predominante involucra consultas filtradas por región específica. Las sucursales típicamente gestionan clientes de su área geográfica, resultando en alta localidad de referencia donde la mayoría de los accesos a CLIENTE se satisfacen con datos del nodo local. Esta localidad mejora rendimiento y reduce tráfico de red entre nodos.

Sin embargo, el esquema presenta limitación en escenarios de análisis centralizado. Consultas de inteligencia empresarial que requieren estadísticas agregadas sobre todos los clientes, como distribución de tipos de clientes a nivel nacional o análisis de patrones de compra cross-regional, necesariamente requieren acceso a todos los fragmentos incurriendo en overhead distribuido. Para estos casos, estrategia complementaria de data warehouse centralizado podría ser apropiada.

La fragmentación de VINO por comunidad autónoma productora presenta trade-off interesante. Favorece operaciones de gestión de inventario local donde delegaciones gestionan predominantemente vinos de su región. Sin embargo, complica operaciones de suministro cross-regional que son inherentes al modelo de negocio donde cualquier cliente puede solicitar cualquier vino. Este trade-off parece aceptable dado que gestión de inventario ocurre más frecuentemente que suministros inter-regionales.

Consistencia y Transacciones Distribuidas

La limitación más significativa del diseño es ausencia de manejo formal de transacciones distribuidas mediante protocolo two-phase commit. Las operaciones que modifican datos en múltiples nodos, como PR_INS_SUMINISTRO que inserta pedido en nodo del cliente y actualiza stock en nodo del vino, no son atómicas en sentido distribuido estricto.

Si la inserción del pedido tiene éxito, pero la actualización de stock falla debido a fallo de red o nodo, el sistema queda en estado inconsistente con pedido registrado, pero stock no decrementado. La recuperación de esta inconsistencia requeriría intervención manual o mecanismos compensatorios no implementados. Para sistema productivo, sería necesario implementar coordinación mediante protocolo 2PC usando características de Oracle como distributed transactions o XA transactions.

Alternativamente, el diseño podría adoptar modelo de consistencia eventual más relajado donde inconsistencias temporales son tolerables y se resuelven mediante procesos de reconciliación periódicos. Este enfoque es común en

sistemas distribuidos de gran escala donde disponibilidad es prioritaria sobre consistencia fuerte. Sin embargo, requeriría re-diseño significativo incluyendo logs de transacciones, procesos de reconciliación y estrategias de resolución de conflictos.

Estrategia de Replicación

La replicación total de PRODUCTOR es decisión pragmática apropiada para tabla pequeña y mayormente estática. Elimina consultas distribuidas frecuentes al validar vinos durante inserciones y consultas. Sin embargo, la implementación mediante inserción manual múltiple es rudimentaria y propensa a inconsistencias.

La prueba de actualización reveló que modificaciones en tabla replicada no se propagan automáticamente, requiriendo actualización manual en todas las réplicas. Para sistema productivo, sería crítico implementar mecanismo automático de propagación. Las opciones incluyen triggers AFTER INSERT, UPDATE, DELETE que ejecuten operaciones remotas mediante database links, o migración a Oracle Materialized Views con refresh ON COMMIT u ON DEMAND que Oracle mantiene automáticamente.

La decisión de no replicar otras tablas parece justificada considerando volúmenes mayores y patrones de acceso localizados. Replicar tablas grandes como REALIZAPEDIDO sería costoso en almacenamiento y complejidad de sincronización sin beneficio significativo porque los accesos están naturalmente localizados por cliente. Sin embargo, replicación selectiva de subconjuntos frecuentemente consultados, como catálogo de vinos popular independientemente de región, podría mejorar rendimiento de consultas de productos.

Transparencia de Distribución

Las vistas globales proporcionan excelente transparencia de distribución desde perspectiva de desarrollador de aplicaciones. Código que consulta v_cliente no requiere conocimiento sobre cómo CLIENTE está fragmentado entre nodos. Esta abstracción facilita mantenimiento porque cambios en esquema de fragmentación no requieren modificar código de aplicación que usa vistas.

Sin embargo, la transparencia es parcial porque operaciones de inserción, actualización y eliminación no pueden dirigirse directamente a vistas globales construidas con UNION. Oracle no permite DML sobre vistas UNION, requiriendo que aplicaciones usen procedimientos almacenados que encapsulan lógica de determinación de nodo apropiado. Esto introduce asimetría donde consultas son transparentes, pero actualizaciones requieren código específico de fragmentación.

Para lograr transparencia completa de actualizaciones, sería necesario implementar vistas actualizables mediante triggers INSTEAD OF que intercepten operaciones DML sobre vistas y las redirijan a fragmentos apropiados. Esta aproximación permitiría código de aplicación como INSERT INTO v_cliente VALUES que sería transparentemente enrutado al fragmento correcto mediante lógica en trigger INSTEAD OF.

Autonomía Local

El diseño respeta bien principio de autonomía local donde cada nodo puede ejecutar operaciones sobre sus datos sin requerir comunicación constante con otros nodos. Las transacciones típicas como registrar pedido de cliente local para vino local se ejecutan completamente en nodo único sin coordinación distribuida.

Sin embargo, la autonomía se compromete en operaciones que requieren validación distribuida. Los triggers que validan restricciones consultando vistas globales, como TR_PK_CLIENTE que verifica unicidad consultando v_cliente, requieren acceso a todos los nodos durante cada validación. Si algún nodo está inaccesible, validaciones fallan imposibilitando inserciones incluso de datos que irían a nodo operativo.

Para mejorar autonomía en presencia de fallos parciales, sería posible implementar validación relajada donde inserción procede si validación en nodos remotos falla con advertencia de verificación pendiente. Procesos de reconciliación posteriores detectarían y resolverían violaciones cuando nodos recuperen conectividad. Este diseño mejora disponibilidad a costa de consistencia estricta, trade-off característico del teorema CAP.

Escalabilidad

El diseño actual es funcionalmente correcto, pero presenta limitaciones de escalabilidad. Las vistas globales con UNION sobre cuatro fragmentos escalan linealmente con número de nodos, significando que agregar nodo quinto requeriría modificar definición de todas las vistas agregando cláusula UNION adicional. Para escenarios con decenas de nodos, este enfoque sería inmanejable.

La ausencia de particionamiento de tablas grandes limita escalabilidad vertical. A medida que tablas como REALIZAPEDIDO crecen con años de historial transaccional, consultas que requieren escaneo completo de fragmento se degradan. Implementar particionamiento por rango de fechas dentro de cada fragmento distribuido permitiría operaciones de poda de particiones mejorando rendimiento de consultas temporales.

El esquema de numeración de códigos no considera distribución. Los códigos de clientes, empleados, vinos y otros son números secuenciales globales que requieren coordinación para garantizar unicidad. En sistema de alta concurrencia con inserciones simultáneas en múltiples nodos, generar códigos únicos requiere mecanismo de coordinación como servicio centralizado de secuencias o asignación de rangos disjuntos por nodo. El diseño actual no especifica estrategia de generación de claves distribuidas.

Gestión de Fallos

El sistema carece de mecanismos explícitos de manejo de fallos de nodos. Si alguna delegación pierde conectividad, las vistas globales fallan porque UNION requiere acceso exitoso a todos los fragmentos. Esta fragilidad compromete disponibilidad porque fallo de nodo único hace sistema completo inoperable para consultas globales.

Para mejorar tolerancia a fallos, sería posible implementar materialized views locales que cachean datos remotos permitiendo operaciones en modo

degradado durante fallos de conectividad. Cuando nodo está inaccesible, consultas operarían sobre snapshot posiblemente desactualizado con advertencia a usuarios. Al recuperar conectividad, refresh de materialized view resincronizaría datos.

La ausencia de logging de transacciones distribuidas complica recuperación ante fallos parciales. Si transacción multi-nodo falla después de modificar algunos nodos, pero no todos, no existe log que permita rollback o replay coordinado. Sistema productivo requeriría write-ahead logs distribuidos y coordinador de recuperación que garantice que transacciones sean atómicas incluso ante fallos.

Seguridad y Control de Acceso

El modelo de privilegios implementado es simplista otorgando permisos idénticos a todos los usuarios sobre todos los objetos. En sistema real, diferentes delegaciones requerirían permisos diferenciados basándose en principio de menor privilegio donde cada nodo accede solo datos necesarios para sus operaciones.

Por ejemplo, delegación de Madrid debería tener permisos completos sobre sus clientes locales, pero solo lectura sobre clientes de otras delegaciones para casos de soporte o análisis. Los procedimientos almacenados que ejecutan operaciones cross-node ejecutarían con privilegios elevados mediante derechos de definer, mientras aplicaciones interactivas tendrían permisos restringidos.

La implementación de seguridad a nivel de fila mediante Oracle Virtual Private Database permitiría políticas sofisticadas donde consultas a vistas globales automáticamente filtran datos basándose en contexto de sesión. Por ejemplo, usuario de delegación Madrid consultando v_cliente vería solo clientes de su región sin requerir filtros explícitos en consulta, implementando seguridad transparente.

Mantenibilidad y Complejidad

La proliferación de cincuenta triggers y quince procedimientos introduce complejidad operacional significativa. La lógica de negocio distribuida entre múltiples triggers interdependientes dificulta razonamiento sobre comportamiento del sistema y diagnóstico de problemas. Modificar una regla de negocio puede requerir actualizar múltiples triggers en múltiples nodos, proceso propenso a errores.

Para mejorar mantenibilidad, sería beneficioso consolidar lógica relacionada en menor número de módulos bien estructurados. Por ejemplo, toda lógica de validación de integridad referencial distribuida podría centralizarse en paquete PL/SQL único con procedimientos específicos por tipo de validación, invocados desde triggers livianos. Esta consolidación facilitaría testing y modificación.

La documentación del código es adecuada con comentarios descriptivos, pero beneficiaría de documentación arquitectónica complementaria explicando flujos de datos entre nodos, diagramas de secuencia para operaciones complejas distribuidas, y matriz de dependencias entre triggers y procedimientos. Esta documentación facilitaría onboarding de nuevos desarrolladores y operadores.

Viabilidad para Producción

El diseño analizado representa excelente ejercicio académico demostrando conceptos fundamentales de bases de datos distribuidas. Sin embargo, requeriría mejoras substanciales para despliegue productivo. Las áreas críticas incluyen implementación de transacciones distribuidas con garantías ACID, mecanismos de replicación automática y sincronización, estrategias de recuperación ante fallos, optimización de rendimiento para escenarios de alto volumen, y herramientas de monitoreo y diagnóstico.

Para implementación productiva, sería recomendable evaluar uso de características nativas de Oracle más avanzadas. Oracle Distributed Database con database links automáticos, Oracle RAC para alta disponibilidad, Oracle GoldenGate para replicación bidireccional, y Oracle Partitioning para manejo de tablas grandes proporcionarían fundamento robusto con menor código custom. Alternativamente, migración a arquitectura de microservicios con base de datos por servicio ofrecería mayor aislamiento y escalabilidad independiente.

CONCLUSIONES

Cumplimiento de Objetivos

El análisis exhaustivo del repositorio de base de datos distribuida ha cumplido satisfactoriamente los objetivos establecidos. Se identificó correctamente el esquema de fragmentación horizontal primaria basado en criterios geográficos aplicado a las tablas principales, así como la fragmentación horizontal derivada para tablas relacionadas. Se comprendieron los mecanismos que garantizan integridad y consistencia mediante cincuenta triggers coordinados y quince procedimientos almacenados que encapsulan lógica distribuida compleja.

La evaluación de transparencia confirmó que las vistas globales proporcionan abstracción efectiva permitiendo consultas sin conocimiento de fragmentación subyacente, aunque con limitación de requerir procedimientos para operaciones de escritura. La autonomía local se verificó observando que operaciones comunes se ejecutan en nodo único, aunque operaciones menos frecuentes requieren coordinación distribuida.

La verificación práctica mediante ejecución de consultas distribuidas demostró funcionamiento correcto del sistema integrando datos de múltiples nodos transparentemente. Las tres consultas de prueba ejecutaron exitosamente produciendo resultados correctos mediante joins, agregaciones y filtros sobre datos distribuidos, validando capacidad del sistema para procesamiento distribuido de consultas.

Aprendizajes Fundamentales

El análisis proporciona entendimiento profundo de principios y desafíos de sistemas de bases de datos distribuidas. El concepto de fragmentación deja de ser abstracto teórico para convertirse en decisión de diseño concreta con trade-offs medibles. La elección entre fragmentación horizontal, vertical o mixta impacta directamente patrones de acceso, rendimiento de consultas y complejidad de mantenimiento.

La replicación emerge como estrategia poderosa con implementación compleja. Si bien mejora disponibilidad y rendimiento de lectura, introduce desafíos significativos de mantenimiento de consistencia y propagación de actualizaciones. La decisión de qué datos replicar requiere análisis cuidadoso de frecuencia de lectura versus escritura y tolerancia a inconsistencia temporal.

La transparencia de distribución es objetivo loable pero parcialmente alcanzable en práctica. Las vistas globales logran transparencia de consulta elegantemente, pero operaciones de actualización requieren lógica específica de fragmentación. Esta asimetría sugiere que sistemas distribuidos reales requieren abstracción en múltiples capas incluyendo capa de aplicación que encapsula complejidad distribuida.

Relevancia del Modelo de Negocio

El caso de estudio de distribuidora vinícola demuestra que bases de datos distribuidas no son solamente construcción académica sino solución práctica para problemas reales de organizaciones geográficamente dispersas. La estructura de cuatro delegaciones regionales con autonomía operativa, pero

coordinación necesaria es patrón común en empresas nacionales o multinacionales.

La fragmentación geográfica alineada con estructura organizacional es estrategia natural que aprovecha localidad inherente de operaciones. Los clientes interactúan predominantemente con sucursales locales, los empleados trabajan en ubicaciones específicas, y los vinos se producen en regiones particulares. Esta localidad natural se traduce directamente en fragmentación eficiente que minimiza comunicación entre nodos.

Sin embargo, el requisito de que cualquier cliente pueda solicitar cualquier vino independientemente de región introduce necesidad inevitable de consultas distribuidas. Este balance entre autonomía local y coordinación global es característico de muchos sistemas empresariales reales, validando relevancia del caso de estudio más allá de contexto académico.

Comparación con Alternativas Arquitectónicas

El análisis permite comparar arquitectura distribuida estudiada con alternativas conceptuales. Una base de datos centralizada sería más simple eliminando complejidad de fragmentación y consultas distribuidas, pero crearía cuello de botella de rendimiento, punto único de fallo y latencia incrementada para delegaciones remotas. El diseño distribuido estudiado ofrece mejor disponibilidad, escalabilidad y rendimiento local a costa de complejidad operacional.

Una arquitectura de microservicios donde cada delegación ejecuta aplicación completa con base de datos privada representaría siguiente nivel de distribución con acoplamiento aún más débil. Esta arquitectura ofrecería máxima autonomía y aislamiento de fallos, pero requeriría mecanismos adicionales para consultas cross-delegation y consistencia eventual, típicamente mediante event sourcing y CQRS.

Arquitectura de data warehouse centralizado complementando bases operacionales distribuidas representaría enfoque híbrido apropiado para organización real. Operaciones transaccionales ejecutan contra bases distribuidas optimizadas para escritura y consultas locales, mientras análisis y reportes ejecutan contra warehouse centralizado actualizado mediante ETL, optimizado para consultas analíticas complejas.

Aplicabilidad de Tecnologías Modernas

El diseño estudiado utiliza características tradicionales de Oracle Database, pero conceptos son aplicables a tecnologías modernas. Bases de datos NoSQL distribuidas como Cassandra, MongoDB, o Couchbase implementan fragmentación automática mediante consistent hashing y replicación configurable. Estas tecnologías priorizan disponibilidad y escalabilidad horizontal sobre consistencia fuerte, trade-off apropiado para muchas aplicaciones web modernas.

Sistemas NewSQL como CockroachDB o Google Spanner combinan escalabilidad distribuida de NoSQL con garantías transaccionales de SQL tradicional, ofreciendo transacciones distribuidas con semántica ACID sobre geografías globales. Estas tecnologías implementan variantes de two-phase

commit optimizadas y replicación síncrona automatizada, resolviendo muchas limitaciones identificadas en diseño analizado.

Plataformas cloud-native como Amazon Aurora o Azure Cosmos DB proporcionan abstracción de distribución donde proveedor gestiona fragmentación, replicación y failover transparentemente. Estas soluciones gestionadas reducen complejidad operacional, pero con trade-off de menos control sobre topología física y mayor costo operativo.

Lecciones para Diseño de Sistemas Distribuidos

El análisis revela principios generalizables para diseño de cualquier sistema distribuido. Primero, fragmentación debe alinearse con patrones de acceso naturales del dominio, aprovechando localidad inherente para minimizar coordinación distribuida. Segundo, transparencia de distribución facilita desarrollo, pero no elimina necesidad de considerar distribución en decisiones de diseño críticas.

Tercero, consistencia, disponibilidad y tolerancia a particiones no son simultáneamente alcanzables plenamente según teorema CAP, requiriendo decisiones explícitas sobre trade-offs aceptables para cada caso de uso. Cuarto, replicación es herramienta poderosa, pero introduce responsabilidad de mantener consistencia entre réplicas mediante estrategias apropiadas.

Quinto, sistemas distribuidos requieren monitoreo, logging y diagnósticos más sofisticados que sistemas centralizados debido a complejidad de comportamiento emergente de múltiples componentes interactuantes. Sexto, testing de sistemas distribuidos debe incluir scenarios de fallo parcial, particiones de red y condiciones de carrera que no existen en sistemas centralizados.

Reflexión sobre el Proceso de Análisis

La metodología de análisis empleada combinando revisión de documentación, estudio de código, ejecución práctica y evaluación crítica demostró efectividad para comprender sistema complejo. La ejecución práctica fue particularmente valiosa porque reveló aspectos no evidentes en lectura de código, como overhead de rendimiento de consultas distribuidas y comportamiento de triggers en escenarios de error.

El proceso de capturar evidencias visuales mediante pantallazos sistemáticos facilitó verificación de comportamiento y proporcionó material para documentación. Esta práctica es recomendable para cualquier análisis de sistema distribuido donde comportamiento puede ser contraintuitivo y difícil de explicar mediante descripción textual únicamente.

La evaluación crítica identificando no solo fortalezas sino también limitaciones y áreas de mejora proporciona valor más allá de simple descripción de sistema existente. Esta perspectiva crítica es esencial para desarrollar criterio de diseño que permite distinguir soluciones robustas de implementaciones frágiles en proyectos futuros.

Conclusión

El sistema de base de datos distribuida analizado constituye implementación sólida de conceptos fundamentales aplicados acaso de uso realista. Demuestra

correcta aplicación de fragmentación horizontal, replicación selectiva, transparencia mediante vistas, y mantenimiento de integridad mediante triggers distribuidos. El sistema funciona correctamente para escenarios diseñados y proporciona base educativa excelente para comprender bases de datos distribuidas.

Sin embargo, el análisis crítico revela que transición de prototipo académico a sistema productivo requeriría mejoras substanciales en gestión de transacciones distribuidas, tolerancia a fallos, escalabilidad, seguridad y herramientas operacionales. Estas limitaciones no disminuyen valor educativo, sino que reflejan realidad de que sistemas distribuidos productivos son genuinamente complejos requiriendo expertos especializada y herramientas sofisticadas.

Para estudiante de bases de datos distribuidas, este análisis proporciona comprensión práctica de cómo conceptos teóricos se materializan en código ejecutable, qué decisiones de diseño son críticas, y qué trade-offs son inherentes a distribución. Este conocimiento práctico complementa fundamentos teóricos preparando para participar efectivamente en proyectos distribuidos del mundo real.

REFERENCIAS BIBLIOGRÁFICAS

1. Özsu, M. T., & Valduriez, P. (2020). *Principles of Distributed Database Systems* (4th ed.). Springer International Publishing.
2. Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Pearson Education.
3. Date, C. J. (2004). *An Introduction to Database Systems* (8th ed.). Addison-Wesley.
4. Oracle Corporation. (2023). *Oracle Database 19c Documentation: Distributed Database Concepts*. https://docs.oracle.com/en/database/oracle/oracle-database/19/admin/distributed-database-concepts.html
5. Oracle Corporation. (2023). *Oracle Database PL/SQL Language Reference*. https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/
6. Brewer, E. A. (2012). "CAP Twelve Years Later: How the 'Rules' Have Changed". *IEEE Computer*, 45(2), 23-29.
7. Gray, J., & Reuter, A. (1993). *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers.
8. Bernstein, P. A., & Newcomer, E. (2009). *Principles of Transaction Processing* (2nd ed.). Morgan Kaufmann.
9. Tanenbaum, A. S., & Van Steen, M. (2017). *Distributed Systems: Principles and Paradigms* (3rd ed.). CreateSpace Independent Publishing Platform.
10. Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media.

ANEXOS

Los scripts SQL completos del sistema, incluyendo cualquier corrección o adaptación realizada durante el proceso de análisis y ejecución, están disponibles en el repositorio de GitHub del estudiante:
URL del Repositorio: https://github.com/Over1185/Tarea-DB