

UNIVERSIDAD MAYOR DE SAN SIMÓN



**CARRERA DE INGENIERÍA
INFORMÁTICA - SISTEMAS**

PROYECTO INMOBILIARIA



PROYECTO FINAL – TEMAS VARIOS

GRUPO: 13

NOMBRES:

- Vera Viscarra Emerson Denis
- Revollo Huanca Andres Juan Jose
- Rojas Becerra Guillermo
- Limachi Cuiza Daniel Boris

MATERIA:

Taller De Base De Datos

DOCENTE:

Lic. Boris Marcelo Calancha Navia

FECHA:

28-06-2023

Cochabamba–Bolivia

INDICE

SEGURIDAD EN BASE DE DATOS

.....	Pág. 1
I. Introducción.....	Pág. 2
II. Archivos LOG	Pág. 3
III. Técnicas de Seguridad	Pág. 4
IV. Conclusiones	Pág. 6

BASE DE DATOS NO SQL

.....	Pág. 7
I. Introducción.....	Pág. 8
II. Diferencias y similitudes entre BD SQL y BD NoSQL ...	Pág. 9
III. Características importantes de las BD NoSQL	Pág. 12
IV. Otros aspectos importantes de las BD NoSQL	Pág. 14
V. Conclusión	Pág. 15

TECNOLOGIAS DE ACCESO A BASE DE DATOS

.....	Pág. 16
I. Introducción.....	Pág. 17
II. ODBC (Open Database Connectivity).....	Pág. 18
III. JDBC (Java Database Connectivity).....	Pág. 19
IV. Conexión nativa en MySQL	Pág. 20
V. Elección del tipo de conexión.....	Pág. 21

DESARROLLO DEL PROTOTIPO

.....	Pág. 22
I. Ambiente de programación	Pág. 23
II. Interfaz de usuario	Pág. 24
III. Conexión a la base de datos.....	Pág. 30
IV. Uso de estructuras e instrucciones de PL/SQL.....	Pág. 33
V. Documentación y explicación de la conexión nativa.....	Pág. 35

RESULTADOS DE PRUEBAS REALIZADAS

.....	Pág. 36
-------	---------

Seguridad en base de datos

La seguridad en las bases de datos es un aspecto fundamental para proteger la integridad, la confidencialidad y la disponibilidad de los datos almacenados. En esta sección, exploraremos diversas técnicas y consideraciones relacionadas con la seguridad en base de datos.

I. Introducción

En la era digital actual, donde la información es un activo valioso, la seguridad de las bases de datos se ha vuelto una preocupación crucial para organizaciones y usuarios por igual. Las bases de datos almacenan y gestionan grandes cantidades de información sensible, como datos personales, financieros o empresariales, lo que las convierte en un objetivo atractivo para los ciberdelincuentes.



La importancia de la seguridad en las bases de datos radica en la necesidad de proteger la confidencialidad, integridad y disponibilidad de la información que contienen. Veamos brevemente algunos de los riesgos potenciales asociados a la falta de seguridad en las bases de datos:

- ❖ Acceso no autorizado: Si las bases de datos no están adecuadamente protegidas, pueden ser vulnerables a accesos no autorizados. Esto podría permitir que personas malintencionadas obtengan información confidencial, realicen modificaciones no autorizadas o incluso eliminen datos importantes.
- ❖ Fugas de datos: Una violación de seguridad en una base de datos puede conducir a la filtración de datos sensibles. Esto puede tener consecuencias graves, como el robo de identidad, el fraude financiero o la pérdida de la confianza de los clientes y socios comerciales.
- ❖ Ataques de inyección de código: Las bases de datos pueden ser víctimas de ataques de inyección de código, donde los atacantes aprovechan vulnerabilidades en las consultas de las bases de datos para ejecutar comandos no deseados. Esto puede llevar a la manipulación de datos, la destrucción de la estructura de la base de datos o incluso al control total del sistema.
- ❖ Pérdida de datos: La falta de medidas adecuadas de respaldo y recuperación puede resultar en la pérdida irreversible de datos en caso de fallos del sistema, desastres naturales o errores humanos. Esto puede tener un impacto devastador en una organización, especialmente si se trata de información crítica o de valor estratégico.

Para mitigar estos riesgos y garantizar la seguridad de las bases de datos, es fundamental implementar medidas de protección adecuadas. Esto implica utilizar métodos de autenticación y control de acceso robustos, encriptar los datos almacenados, realizar copias de seguridad periódicas y mantenerse al día con las actualizaciones de seguridad del software utilizado.

II. Archivos LOG

Los archivos de registro (LOG) son componentes fundamentales de las bases de datos que cumplen una función crucial en la gestión de transacciones y la seguridad de los datos. Estos archivos se utilizan para registrar y almacenar de forma secuencial todas las actividades y cambios realizados en la base de datos, lo que permite realizar un seguimiento detallado de las operaciones realizadas.

La función principal de los archivos LOG es proporcionar una capa adicional de seguridad y respaldo para las bases de datos. Cada vez que se realiza una transacción o se modifica un registro en la base de datos, los detalles de dicha operación se registran en el archivo LOG correspondiente. Estos registros incluyen información como la identificación de la transacción, las tablas y registros afectados, los valores modificados y cualquier otra acción relevante.



La principal utilidad de los archivos LOG radica en la recuperación ante fallas. En caso de que ocurra un fallo del sistema, como un corte de energía o un error de hardware, los archivos LOG permiten reconstruir la base de datos y restaurarla a un estado consistente. Al tener un registro completo de todas las transacciones realizadas, se puede volver a aplicar o deshacer las operaciones registradas en el archivo LOG, lo que garantiza que la base de datos se restaure a un estado válido y coherente.

Además de su función en la recuperación ante fallas, los archivos LOG también desempeñan un papel importante en la auditoría de la base de datos. Al registrar todas las transacciones y cambios realizados, los archivos LOG proporcionan un historial detallado de las actividades de la base de datos. Esto permite realizar análisis de seguridad y auditorías para identificar posibles irregularidades, detectar intentos de acceso no autorizado o analizar el flujo de datos dentro del sistema.

Es importante destacar que los archivos LOG deben mantenerse de manera segura y protegida, ya que contienen información crítica y confidencial. Además, es recomendable establecer políticas de retención y respaldo de los archivos LOG para garantizar su disponibilidad a largo plazo y su uso efectivo en caso de necesidad de recuperación o auditoría.

III. Técnicas de Seguridad

La seguridad en las bases de datos es un aspecto crucial para proteger la información confidencial y garantizar la integridad de los datos. A continuación, se presentan algunas técnicas y prácticas recomendadas que se pueden implementar para fortalecer la seguridad en las bases de datos:

Control de acceso

El control de acceso es una técnica fundamental que permite limitar quién puede acceder a la base de datos y qué acciones pueden realizar. Algunas medidas de control de acceso incluyen:

- Asignar roles y privilegios específicos a los usuarios según sus responsabilidades.
- Implementar autenticación de dos factores para verificar la identidad de los usuarios.
- Utilizar mecanismos de inicio de sesión seguros, como contraseñas robustas y políticas de cambio periódico de contraseñas.
- Aplicar restricciones de acceso a nivel de objeto, como tablas o vistas, para limitar las operaciones que los usuarios pueden realizar en determinados datos.

Autenticación y autorización de usuarios

La autenticación se refiere a la verificación de la identidad de un usuario antes de permitirle acceder a la base de datos. La autorización, por otro lado, consiste en otorgar los permisos necesarios para que un usuario acceda y realice acciones específicas en la base de datos. Algunas prácticas para mejorar la autenticación y autorización incluyen:

- Utilizar métodos de autenticación sólidos, como contraseñas seguras, autenticación biométrica o autenticación de dos factores.
- Implementar políticas de autorización basadas en roles, donde los permisos se asignan a los roles y los usuarios se asignan a esos roles.
- Revisar y actualizar regularmente los privilegios de usuario para garantizar que se ajusten a las necesidades y responsabilidades actuales.

Encriptación de datos

La encriptación es una técnica eficaz para proteger la confidencialidad de los datos almacenados en una base de datos. Algunas formas de implementar la encriptación en una base de datos incluyen:

- Encriptación de datos en reposo: se cifran los archivos y registros de la base de datos cuando están almacenados en el disco.
- Encriptación de datos en tránsito: se utiliza protocolos seguros, como SSL, para proteger la comunicación entre la base de datos y las aplicaciones que acceden a ella.
- Encriptación de columna: se encriptan datos específicos en una columna de la base de datos, lo que proporciona una capa adicional de seguridad para información sensible.

Monitoreo de actividad sospechosa

El monitoreo constante de la actividad en la base de datos es esencial para identificar posibles amenazas o comportamientos anómalos. Algunas prácticas de monitoreo incluyen:

- Implementar registros de auditoría y archivos de registro (LOG) para registrar las transacciones y cambios realizados en la base de datos.
- Establecer alertas y notificaciones para detectar actividades sospechosas, como intentos de acceso no autorizado o cambios inusuales en los datos.
- Utilizar sistemas de detección de intrusiones (IDS) o sistemas de prevención de intrusiones (IPS) para monitorear y bloquear intentos de ataques maliciosos.

Prácticas recomendadas adicionales

Además de las técnicas mencionadas, algunas prácticas adicionales que pueden fortalecer la seguridad en las bases de datos incluyen:

- Mantener el software de la base de datos actualizado con los últimos parches y actualizaciones de seguridad.
- Realizar copias de seguridad periódicas y almacenarlas de forma segura en ubicaciones fuera de la base de datos principal.
- Limitar la exposición de la base de datos en la red, utilizando firewalls y segmentación de red para protegerla de ataques externos.
- Capacitar a los usuarios y al personal de la base de datos en buenas prácticas de seguridad, como la protección de contraseñas y la detección de ataques de ingeniería social.

Implementar estas técnicas de seguridad en las bases de datos ayudará a proteger la información confidencial, prevenir ataques y asegurar la integridad de los datos almacenados. Es importante tener en cuenta que la seguridad en las bases de datos es un proceso continuo y que se deben mantener actualizadas las prácticas de seguridad para hacer frente a las nuevas amenazas y vulnerabilidades que puedan surgir.



IV. Conclusiones

En conclusión, la seguridad en las bases de datos es de vital importancia para proteger la información sensible y garantizar la integridad de los datos almacenados. A lo largo de este artículo, hemos abordado diversos aspectos relacionados con la seguridad en bases de datos y destacado la importancia de implementar medidas sólidas de seguridad, así como la implementación de los archivos de registro. Algunos puntos clave que resaltamos son los siguientes:

Importancia de la seguridad en bases de datos:

La seguridad en las bases de datos es esencial para proteger la información sensible y garantizar la confianza de los usuarios. Los riesgos potenciales asociados a la falta de seguridad hacen imperativo que las organizaciones adopten medidas de protección adecuadas y estén al tanto de las mejores prácticas en seguridad de bases de datos. Solo así se podrá enfrentar de manera efectiva la creciente amenaza de los ciberataques y salvaguardar los datos críticos para el éxito y la reputación de una organización.

Importancia de archivos de registro (LOG):

Los archivos de registro (LOG) desempeñan un papel vital en las bases de datos al registrar todas las transacciones y cambios realizados. Estos archivos son fundamentales para la recuperación ante fallas y la auditoría de la base de datos, ya que permiten reconstruir y restaurar la base de datos a un estado consistente, así como proporcionar un historial detallado de las actividades realizadas.

Implementación de medidas sólidas de seguridad:

La seguridad en las bases de datos es un aspecto crítico para cualquier organización que maneje información confidencial. Implementar técnicas sólidas de seguridad, mantenerse actualizado y promover una cultura de seguridad son elementos clave para proteger la información y garantizar la confianza y la integridad en el manejo de datos.



Bases de datos NoSQL

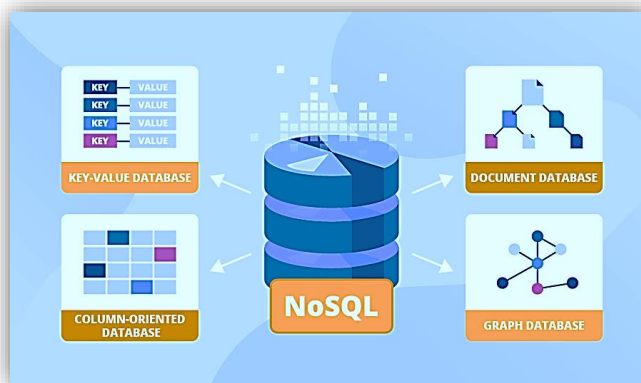
Las bases de datos NoSQL son sistemas de gestión de bases de datos que difieren en su enfoque y estructura de las bases de datos relacionales tradicionales. En esta sección, exploraremos los conceptos y características fundamentales de las bases de datos NoSQL.

I. Introducción

En el mundo de la gestión de datos, las bases de datos NoSQL (Not Only SQL) han ganado popularidad en los últimos años como una alternativa a las bases de datos SQL tradicionales. Estas bases de datos se han desarrollado para abordar las limitaciones y desafíos que surgen al tratar con grandes volúmenes de datos no estructurados o semiestructurados, como documentos, gráficos, series de tiempo y datos enlazados.



A diferencia de las bases de datos SQL, que se basan en el modelo relacional y utilizan tablas con filas y columnas para almacenar datos, las bases de datos NoSQL adoptan diferentes modelos de datos y esquemas flexibles. No siguen el enfoque de tablas relacionales, sino que emplean una variedad de estructuras de datos como documentos, grafos, columnas amplias y pares clave-valor. Esta flexibilidad en los modelos de datos permite un almacenamiento y recuperación eficiente de datos no estructurados y semiestructurados, así como una mayor escalabilidad horizontal.



Las bases de datos NoSQL son especialmente adecuadas en situaciones donde se requiere una alta velocidad de escritura y lectura de datos, escalabilidad horizontal fácil, tolerancia a fallos y la capacidad de trabajar con grandes volúmenes de datos no estructurados. Son ampliamente utilizadas en aplicaciones web y móviles, análisis de Big Data, Internet de las cosas (IoT) y aplicaciones que manejan flujos de datos en tiempo real.

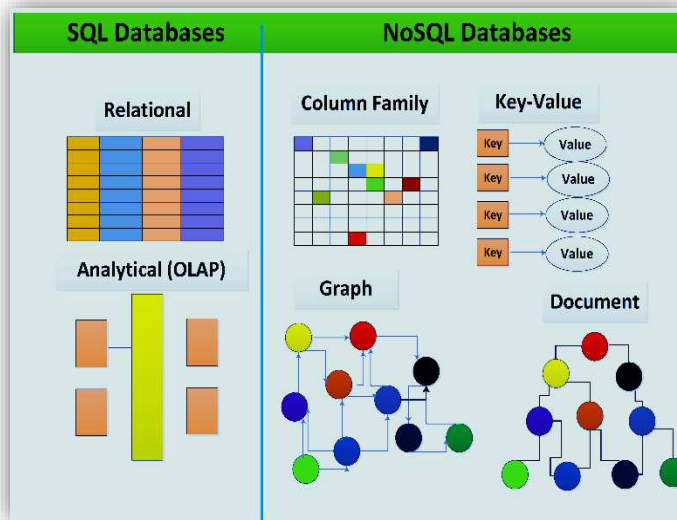
II. Diferencias y similitudes entre BD SQL y BD NoSQL

Las bases de datos SQL y las bases de datos NoSQL son diferentes en varios aspectos, pero también comparten algunas similitudes. A continuación, se detallan las diferencias y similitudes clave entre estos dos tipos de bases de datos:



Modelo de datos:

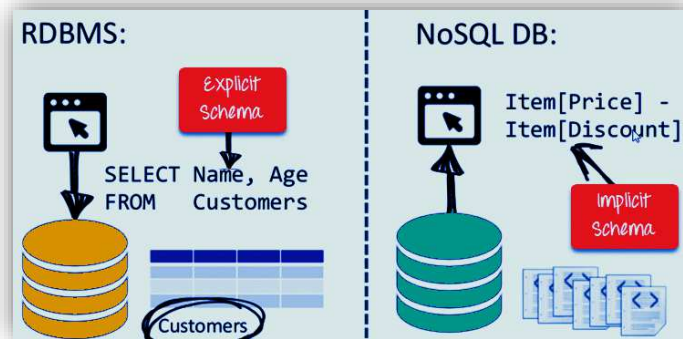
Bases de datos SQL: Utilizan un modelo relacional en el que los datos se organizan en tablas con filas y columnas. La relación entre las tablas se establece mediante claves primarias y claves externas.



Bases de datos NoSQL: Utilizan diversos modelos de datos, como documentos, grafos, columnas amplias o pares clave-valor. Cada modelo se adapta mejor a diferentes tipos de datos y permite una mayor flexibilidad en el esquema.

Lenguaje de consulta:

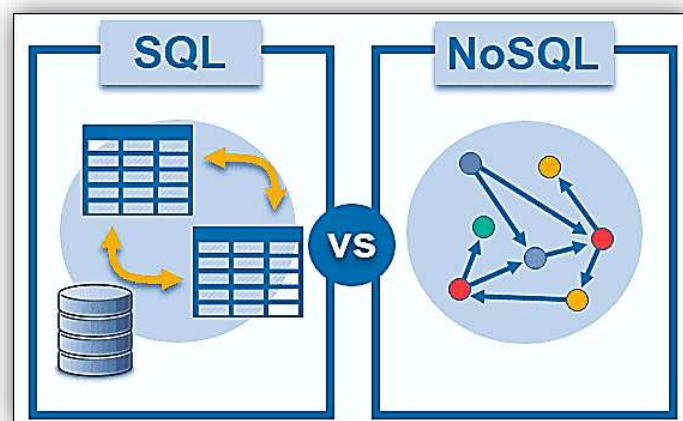
Bases de datos SQL: Utilizan SQL (Structured Query Language) como lenguaje estándar para consultar y manipular los datos almacenados. SQL proporciona una sintaxis y una semántica bien definidas para realizar consultas y operaciones en bases de datos relacionales.



Bases de datos NoSQL: No todas las bases de datos NoSQL utilizan un lenguaje de consulta estándar como SQL. Algunas bases de datos NoSQL tienen sus propios lenguajes de consulta específicos o utilizan interfaces de programación de aplicaciones (API) para acceder y manipular los datos.

Escalabilidad:

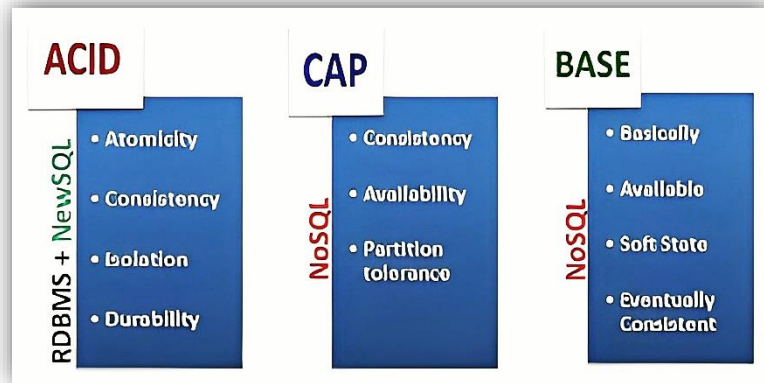
Bases de datos SQL: Generalmente, la escalabilidad en bases de datos SQL se logra mediante la replicación y la partición de datos en múltiples servidores. Sin embargo, la escalabilidad horizontal puede ser más compleja debido a las restricciones del modelo relacional.



Bases de datos NoSQL: Suelen ser altamente escalables, especialmente en entornos distribuidos. Pueden escalar horizontalmente fácilmente mediante la adición de más nodos a la infraestructura, lo que permite manejar grandes volúmenes de datos y cargas de trabajo intensivas.

Consistencia:

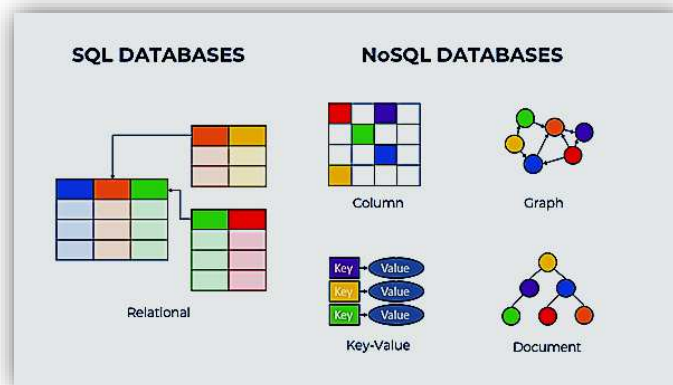
Bases de datos SQL: Siguen el principio ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), lo que significa que garantizan una alta consistencia de los datos en todas las operaciones. Las transacciones en bases de datos SQL se llevan a cabo de manera atómica y aseguran la integridad de los datos.



Bases de datos NoSQL: La consistencia puede variar dependiendo del modelo de consistencia elegido por la base de datos NoSQL. Algunas bases de datos NoSQL priorizan la disponibilidad y la partición tolerante a fallos (AP), lo que puede resultar en una consistencia eventual en lugar de una consistencia estricta.

Flexibilidad en el esquema:

Bases de datos SQL: Utilizan un esquema rígido y estructurado, donde se define la estructura de las tablas y se requiere que los datos cumplan con dicho esquema. Cualquier cambio en el esquema puede ser complicado y requerir modificaciones en la estructura existente.



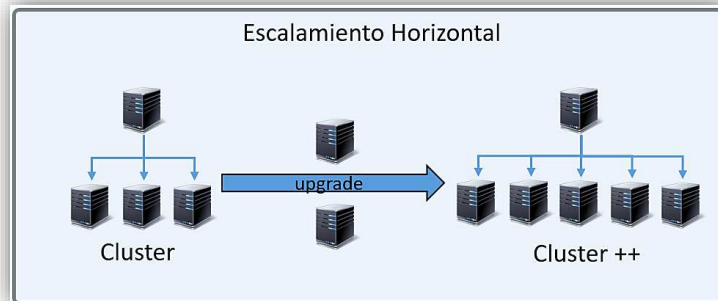
Bases de datos NoSQL: Ofrecen una mayor flexibilidad en el esquema, lo que significa que no se requiere una estructura fija y los datos pueden tener diferentes estructuras en una misma base de datos. Esto permite una fácil adaptación a cambios en los requisitos y una mayor agilidad en el desarrollo de aplicaciones.

III. Características importantes de las BD NoSQL

Las bases de datos NoSQL (Not Only SQL) se han vuelto cada vez más populares debido a su capacidad para manejar grandes volúmenes de datos no estructurados y semiestructurados de manera eficiente. A continuación, se presentan algunas características importantes que definen a las bases de datos NoSQL:

Escalabilidad horizontal

Una de las principales ventajas de las bases de datos NoSQL es su capacidad para escalar horizontalmente de manera eficiente. Esto significa que se pueden agregar fácilmente nuevos servidores o nodos a la infraestructura para manejar aumentos en el volumen de datos y las cargas de trabajo. Al permitir una distribución de datos en múltiples servidores, las bases de datos NoSQL pueden soportar aplicaciones que requieren un alto rendimiento y una capacidad de almacenamiento escalable.



Flexibilidad en el esquema

A diferencia de las bases de datos SQL tradicionales, que tienen un esquema rígido y estructurado, las bases de datos NoSQL ofrecen flexibilidad en el esquema. Esto significa que no se requiere una estructura fija para los datos y no es necesario definir previamente las columnas y tablas antes de almacenar los datos. Esto facilita la incorporación de nuevos tipos de datos y cambios en el esquema sin requerir modificaciones en la estructura existente.



Distribución geográfica

Las bases de datos NoSQL están diseñadas para funcionar en entornos distribuidos, lo que permite la distribución geográfica de los datos. Esto significa que los datos pueden almacenarse y replicarse en diferentes ubicaciones geográficas, lo que brinda beneficios como la alta disponibilidad, la tolerancia a fallos y un mejor rendimiento para los usuarios ubicados en diferentes regiones.

Almacenamiento de datos no estructurados y semiestructurados

Las bases de datos NoSQL son ideales para almacenar y trabajar con datos no estructurados y semiestructurados. Pueden manejar documentos, gráficos, columnas amplias y pares clave-valor, lo que les permite ser utilizadas en aplicaciones que manejan datos de diferentes formatos y estructuras. Esto es especialmente útil en escenarios como el análisis de Big Data, donde los datos pueden tener una variedad de estructuras y esquemas.

Alta velocidad de lectura y escritura

Las bases de datos NoSQL están diseñadas para proporcionar un alto rendimiento en operaciones de lectura y escritura. Debido a su estructura optimizada y su capacidad de escalar horizontalmente, pueden manejar grandes volúmenes de operaciones de lectura y escritura simultáneas, lo que las hace adecuadas para aplicaciones que requieren respuestas rápidas y tiempos de procesamiento cortos.



Modelos de consistencia flexibles

Las bases de datos NoSQL ofrecen diferentes modelos de consistencia, lo que permite ajustar el equilibrio entre consistencia y disponibilidad según las necesidades del proyecto. Algunos modelos de consistencia, como la consistencia eventual, permiten una mayor disponibilidad y tolerancia a fallos, mientras que otros modelos, como la consistencia fuerte, garantizan una consistencia estricta en todos los nodos de la base de datos.

IV. Otros aspectos importantes de las BD NoSQL

Además de las características previamente mencionadas, hay varios aspectos adicionales que son importantes considerar al trabajar con bases de datos NoSQL. A continuación, exploraremos algunos de ellos:

Rendimiento:

Las bases de datos NoSQL están diseñadas para ofrecer un alto rendimiento en operaciones de lectura y escritura. Debido a su estructura optimizada y su capacidad para escalar horizontalmente, pueden manejar grandes volúmenes de datos y realizar operaciones de manera eficiente. Esto las hace especialmente adecuadas para aplicaciones que requieren tiempos de respuesta rápidos y un alto rendimiento en el procesamiento de datos.

Seguridad:

La seguridad de los datos es un aspecto crítico en cualquier sistema de bases de datos. Las bases de datos NoSQL ofrecen medidas de seguridad para proteger los datos almacenados. Esto puede incluir autenticación de usuarios, control de acceso granular y encriptación de datos en reposo y en tránsito. Es importante implementar buenas prácticas de seguridad y seguir los lineamientos de cada base de datos NoSQL específica para garantizar la integridad y confidencialidad de los datos.

Operaciones de lectura y escritura:

Las bases de datos NoSQL están diseñadas para manejar operaciones de lectura y escritura de manera eficiente. Algunas bases de datos NoSQL ofrecen características como escrituras masivas en paralelo, lo que permite realizar múltiples operaciones de escritura simultáneamente. Además, las bases de datos NoSQL generalmente tienen un modelo de datos flexible que permite optimizar las consultas para lograr un rendimiento óptimo en diferentes tipos de operaciones.

Estrategias de replicación:

La replicación es una técnica utilizada en bases de datos NoSQL para mejorar la disponibilidad y la tolerancia a fallos. Al replicar los datos en múltiples servidores, se puede garantizar que los datos estén disponibles incluso en caso de fallos o caídas de servidores individuales. Las bases de datos NoSQL ofrecen diferentes estrategias de replicación, como replicación síncrona o asíncrona, y permiten configurar la cantidad de réplicas y la consistencia requerida.









Soporte de consultas complejas:

Aunque las bases de datos NoSQL no utilizan SQL como lenguaje de consulta estándar, muchas de ellas proporcionan mecanismos para realizar consultas complejas y realizar operaciones de análisis de datos. Algunas bases de datos NoSQL ofrecen

lenguajes de consulta propietarios o extensiones de SQL para facilitar la realización de consultas complejas y operaciones de agregación.

Ejemplos populares de Bases de Datos NoSQL:

Existen numerosas bases de datos NoSQL populares utilizadas en diferentes casos de uso. Algunos ejemplos incluyen:

Type	Example
Key-Value Store	 
Wide Column Store	 
Document Store	 
Graph Store	 

Redis: Una base de datos en memoria que ofrece almacenamiento de pares clave-valor de alto rendimiento.

Cassandra: Una base de datos de columnas amplias diseñada para ofrecer alta disponibilidad y escalabilidad lineal.

MongoDB: Una base de datos de documentos que proporciona almacenamiento flexible de datos no estructurados.

Neo4j: Una base de datos de grafos que permite almacenar y consultar datos relacionales en forma de grafos.

V. Conclusión

En esta sección, hemos explorado las bases de datos NoSQL y hemos destacado sus diferencias y similitudes con las bases de datos SQL tradicionales. Hemos descrito las características importantes de las bases de datos NoSQL, como la escalabilidad horizontal, la flexibilidad en el esquema, la distribución geográfica y la capacidad de almacenar datos no estructurados y semiestructurados. También hemos discutido otros aspectos relevantes, como el rendimiento, la seguridad, las operaciones de lectura y escritura, las estrategias de replicación y el soporte de consultas complejas.

Es importante tener en cuenta que la elección entre una base de datos SQL y una base de datos NoSQL dependerá de los requisitos específicos del proyecto. Las bases de datos NoSQL son ideales para aplicaciones que manejan grandes volúmenes de datos no estructurados y requieren una alta escalabilidad y flexibilidad en el esquema. Por otro lado, las bases de datos SQL son más adecuadas para aplicaciones que necesitan un modelo de datos estructurado y una fuerte consistencia en las operaciones.

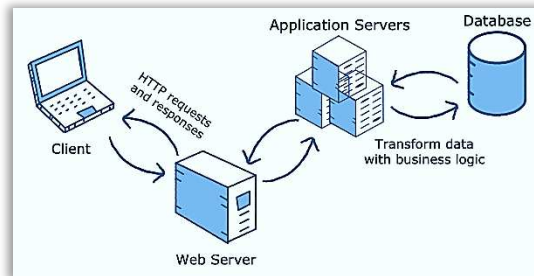
Al elegir una base de datos NoSQL, es crucial evaluar cuidadosamente los requisitos y las necesidades del proyecto. Esto incluye considerar factores como el rendimiento, la seguridad, la escalabilidad, la facilidad de uso y el soporte de consultas complejas. Además, es recomendable investigar y familiarizarse con las diferentes bases de datos NoSQL disponibles en el mercado, como MongoDB, Cassandra, Neo4j y Redis, para encontrar aquella que mejor se adapte a las necesidades del proyecto.

Tecnologías de acceso a bases de datos

En esta sección, investigaremos las tecnologías de acceso a bases de datos ampliamente utilizadas y hablaremos en dos en concreto: ODBC (Open Database Connectivity) y JDBC (Java Database Connectivity). También analizaremos si la conexión nativa en MySQL requiere intermediarios y justificaremos nuestra elección para esta fase del proyecto.

I. Introducción

La conexión a la base de datos es vital para acceder, almacenar y manipular datos de manera segura y eficiente. Es un componente esencial en el desarrollo de aplicaciones que interactúan con bases de datos y desempeña un papel fundamental en la funcionalidad, el rendimiento y la integridad de los datos de la aplicación.



En el mundo de la programación y el desarrollo de aplicaciones, existen diversas tecnologías de acceso a bases de datos que son ampliamente utilizadas debido a su eficiencia, versatilidad y compatibilidad con diferentes sistemas de gestión de bases de datos. A continuación, se mencionan algunas de estas tecnologías:

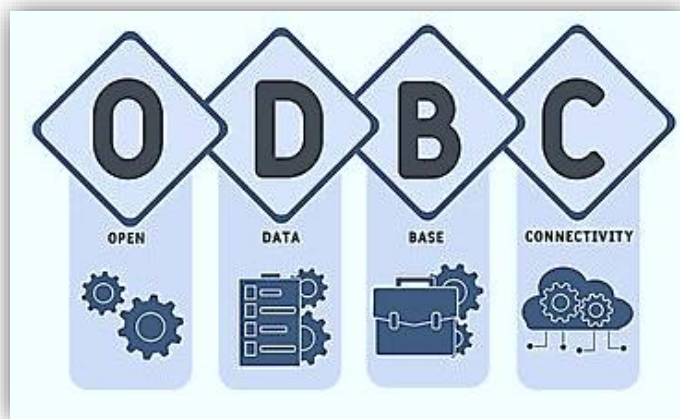
- ❖ **Hibernate:** Hibernate es un framework de mapeo objeto-relacional para Java que facilita el acceso y la manipulación de datos en bases de datos relacionales. Proporciona una capa de abstracción que permite a los desarrolladores trabajar con objetos en lugar de escribir consultas SQL directamente.
- ❖ **Entity Framework:** Entity Framework es un ORM (Object-Relational Mapping) desarrollado por Microsoft para la plataforma .NET. Permite a los desarrolladores trabajar con bases de datos relacionales utilizando objetos y consultas LINQ (Language Integrated Query).
- ❖ **Django ORM:** Django ORM es un módulo de mapeo objeto-relacional incorporado en el framework de desarrollo web Django, utilizado en Python. Proporciona una forma sencilla y eficiente de interactuar con bases de datos relacionales utilizando modelos y consultas basadas en objetos.
- ❖ **MongoDB:** MongoDB es una base de datos NoSQL ampliamente utilizada que utiliza un modelo de documentos en lugar de un modelo relacional. Proporciona un acceso flexible y eficiente a los datos y es especialmente adecuada para aplicaciones con datos no estructurados o que requieren una alta escalabilidad.
- ❖ **Firebase Realtime Database:** Firebase Realtime Database es una base de datos en tiempo real alojada en la nube, proporcionada por Google. Permite el acceso en tiempo real a los datos y es ampliamente utilizada en aplicaciones web y móviles que requieren sincronización instantánea de datos entre diferentes dispositivos.

Estas son solo algunas de las tecnologías de acceso a bases de datos ampliamente utilizadas en el desarrollo de aplicaciones, Pero ahora hablaremos de tres en particular.

II. ODBC (Open Database Connectivity)

ODBC (Open Database Connectivity) es una tecnología que proporciona una interfaz estándar para acceder y administrar bases de datos desde aplicaciones. Fue desarrollado por Microsoft en la década de 1990 y se ha convertido en un estándar de facto en la industria de la programación para el acceso a bases de datos relacionales.

La principal ventaja de ODBC es que actúa como un intermediario entre la aplicación y la base de datos, permitiendo que la aplicación se comunique con diferentes sistemas de gestión de bases de datos (SGBD) utilizando un conjunto común de funciones. Esto significa que una aplicación desarrollada con ODBC puede acceder a múltiples bases de datos sin tener que cambiar su código, siempre que se disponga de un controlador ODBC adecuado para el SGBD específico.



Una arquitectura típica de ODBC implica tres componentes principales: la aplicación, el controlador ODBC y el controlador del SGBD. La aplicación utiliza las funciones ODBC para enviar consultas y recibir resultados desde el controlador ODBC. A su vez, el controlador ODBC traduce estas consultas a un formato que el controlador del SGBD pueda entender y se comunica con el SGBD para realizar las operaciones requeridas. Los resultados se devuelven a la aplicación a través del controlador ODBC.

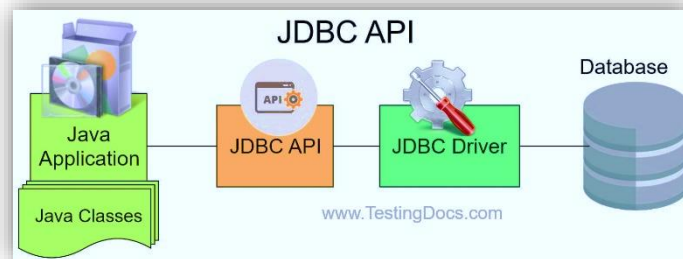
Una de las fortalezas de ODBC es su capacidad de ser utilizado en diferentes lenguajes de programación y plataformas. Existen controladores ODBC disponibles para una amplia gama de lenguajes, incluyendo C/C++, Java, Python, PHP, entre otros. Esto permite a los desarrolladores utilizar ODBC en su lenguaje preferido y acceder a bases de datos compatibles con ODBC sin problemas de compatibilidad.

Cuando se trata de conectarse a MySQL utilizando ODBC, existen controladores ODBC específicos proporcionados por varios proveedores, como MySQL Connector/ODBC. Estos controladores permiten la conexión y el acceso a bases de datos de MySQL desde una aplicación utilizando ODBC. Al utilizar ODBC para acceder a MySQL, es importante tener en cuenta consideraciones como la seguridad, el rendimiento y la configuración adecuada de los controladores y las conexiones.

III. JDBC (Java Database Connectivity)

JDBC (Java Database Connectivity) es una API estándar de Java que proporciona un conjunto de interfaces y clases para interactuar con bases de datos relacionales. Fue desarrollado por Oracle y se ha convertido en un componente fundamental para acceder a bases de datos desde aplicaciones Java.

JDBC permite a las aplicaciones Java establecer conexiones con bases de datos, enviar consultas y recibir resultados, y administrar transacciones. Proporciona una capa de abstracción que permite a los desarrolladores interactuar con diferentes bases de datos mediante un conjunto común de métodos y clases.



La integración de JDBC en la plataforma Java es nativa, lo que significa que está disponible de forma predeterminada sin necesidad de instalar bibliotecas adicionales. Las clases e interfaces de JDBC se encuentran en el paquete `java.sql` y se utilizan en conjunto con controladores de bases de datos específicos (también conocidos como controladores JDBC) para establecer la comunicación entre la aplicación Java y la base de datos.

Para utilizar JDBC con MySQL, es necesario obtener el controlador JDBC específico para MySQL, que es proporcionado por el proyecto MySQL Connector/J de Oracle. Este controlador se encarga de establecer la conexión con la base de datos MySQL y de traducir las consultas SQL enviadas desde la aplicación Java al formato comprensible por MySQL. Algunos de los beneficios de utilizar JDBC en el contexto de MySQL son los siguientes:

- ❖ **Portabilidad:** JDBC es una API estándar de Java, lo que significa que las aplicaciones desarrolladas utilizando JDBC son portables y pueden ejecutarse en cualquier plataforma compatible con Java.
- ❖ **Rendimiento:** JDBC proporciona un acceso eficiente a la base de datos MySQL, permitiendo enviar consultas, recibir resultados y administrar transacciones de manera optimizada.
- ❖ **Seguridad:** JDBC ofrece mecanismos de seguridad para establecer conexiones seguras con MySQL, utilizando técnicas como el cifrado de datos y la autenticación de usuarios.
- ❖ **Escalabilidad:** JDBC permite administrar transacciones de manera efectiva, lo que es especialmente útil en aplicaciones que requieren operaciones complejas y transaccionales en la base de datos.

IV. Conexión nativa en MySQL

En MySQL, existe la posibilidad de utilizar una conexión nativa que permite establecer una comunicación directa entre la aplicación y la base de datos sin necesidad de intermediarios adicionales. MySQL proporciona controladores y protocolos nativos que permiten una conectividad eficiente y optimizada para acceder a la base de datos.

Una conexión nativa en MySQL se basa en el uso del protocolo de red de MySQL, que se encarga de la comunicación entre la aplicación y el servidor de MySQL. Este protocolo permite una interacción de alto rendimiento y baja latencia, lo que resulta en un acceso rápido a la base de datos.



Una ventaja significativa de utilizar una conexión nativa es que no se requiere ningún intermediario adicional, como ODBC o JDBC. Esto significa que no se necesita un controlador adicional o una API específica para establecer la conexión y enviar consultas a MySQL. En lugar de eso, se puede utilizar directamente la biblioteca y las funciones proporcionadas por MySQL para comunicarse con la base de datos.

Al utilizar una conexión nativa, se pueden aprovechar las ventajas específicas de MySQL, como su rendimiento optimizado y su soporte para características avanzadas de bases de datos. Además, al eliminar la necesidad de intermediarios, se puede lograr una mayor eficiencia y simplicidad en el acceso a la base de datos.

Sin embargo, también existen algunas consideraciones al utilizar una conexión nativa en MySQL. A diferencia de tecnologías como ODBC y JDBC, que son estándares y se pueden utilizar con una variedad de bases de datos, una conexión nativa solo es compatible con MySQL. Esto implica que, si se desea cambiar a otro SGBD, puede requerir modificaciones en el código y adaptaciones adicionales.

Además, al utilizar una conexión nativa, se está atado a las características y funcionalidades específicas de MySQL. Si se requiere compatibilidad con múltiples bases de datos o se necesita acceder a características específicas de otros SGBD, puede ser más conveniente utilizar tecnologías como ODBC o JDBC, que ofrecen una capa de abstracción más amplia y son más portables.

V. Elección del tipo de conexión

En esta fase del proyecto, hemos seleccionado utilizar JDBC en conjunto con MySQL. A continuación, justificaremos nuestra elección teniendo en cuenta los requisitos y objetivos específicos del proyecto.

Uno de los factores clave que ha influido en nuestra elección es que todos los integrantes del equipo tienen un conocimiento sólido y están familiarizados con el lenguaje de programación Java. Java es un lenguaje ampliamente utilizado y cuenta con una gran comunidad de desarrolladores, lo que nos brinda una amplia gama de recursos y soporte.



En cuanto a otros factores para la elección de JDBC tenemos los siguientes:

- ❖ **Compatibilidad con MySQL:** JDBC es una tecnología estándar y ampliamente utilizada para acceder a bases de datos relacionales en el entorno Java. Es compatible con múltiples SGBD, incluido MySQL, lo que nos permite establecer una conexión directa y eficiente con la base de datos.
- ❖ **Integración con Java:** JDBC es una API de Java, lo que significa que se integra de manera nativa con el lenguaje de programación Java. La familiaridad con Java nos permite aprovechar al máximo las capacidades y características del lenguaje, así como las herramientas y bibliotecas asociadas. Esto facilita el desarrollo, la depuración y el mantenimiento del código, ya que todos los miembros del equipo pueden colaborar y contribuir de manera efectiva.
- ❖ **Rendimiento y eficiencia:** JDBC proporciona un acceso eficiente a la base de datos MySQL, lo que nos permite enviar consultas, recibir resultados y administrar transacciones de manera optimizada. Esto es especialmente importante para cumplir con los requisitos de rendimiento del proyecto.
- ❖ **Portabilidad:** JDBC es una API estándar de Java, lo que significa que las aplicaciones desarrolladas utilizando JDBC son portables y pueden ejecutarse en cualquier plataforma compatible con Java. Esto nos brinda flexibilidad y facilita la adaptación del prototipo a diferentes entornos.

La elección de utilizar JDBC en conjunto con MySQL se justifica por su compatibilidad que tiene con éste, su integración nativa con Java, su rendimiento y eficiencia, y su portabilidad. Esta tecnología nos permite cumplir con los requisitos específicos del proyecto, como la capacidad de ingresar los datos del cliente y la lista de requerimientos generales y específicos.

Desarrollo del prototipo

En esta sección, se abordará el desarrollo de un prototipo que cumpla con los requerimientos específicos de ingresar datos del cliente y requerimientos en una base de datos. Los siguientes subtítulos se enfocarán en los aspectos clave de la implementación.

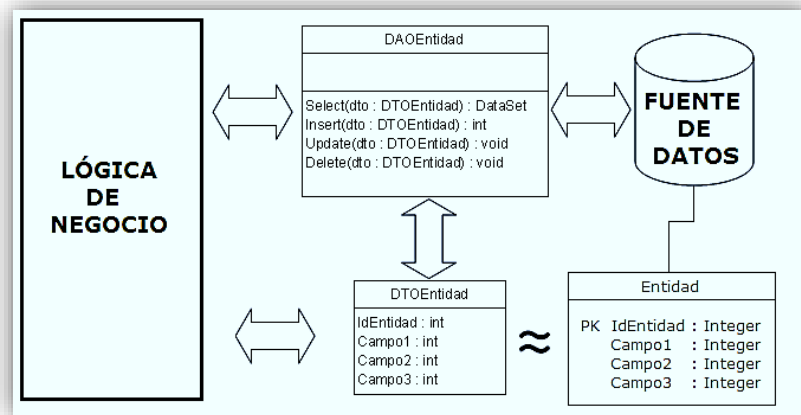
I. Ambiente de programación

La selección del ambiente de programación para el desarrollo del prototipo implica tener en cuenta el lenguaje de programación y las herramientas que mejor se adapten a las necesidades del proyecto. En este caso, dado que se utilizará el lenguaje de programación Java y se desea aprovechar el patrón DAO, hay varias consideraciones que podemos tener en cuenta.

DAO, que significa Data Access Object, es un patrón de diseño de software que se utiliza para abstraer y encapsular la lógica de acceso a datos de la aplicación. Proporciona una capa de abstracción entre la capa de negocio y la capa de acceso a datos, lo que permite separar las operaciones de acceso a la base de datos de la lógica de negocio. Esto facilita el mantenimiento, la escalabilidad y la reutilización del código.

En conjunto, el patrón DAO y Java trabajan para facilitar el acceso y la manipulación de los datos en MySQL. El patrón DAO define una interfaz común para las operaciones de acceso a datos, como consultas y actualizaciones, y proporciona implementaciones concretas que utilizan JDBC para comunicarse con la base de datos MySQL. Esto permite que la lógica de negocio en la aplicación Java se mantenga separada de los detalles de la base de datos, lo que mejora la mantenibilidad y modularidad del código.

La implementación del patrón DAO en Java generalmente implica la creación de interfaces DAO que definen los métodos de acceso a datos y las clases DAO concretas que implementan esos métodos utilizando JDBC para interactuar con MySQL. Estas clases DAO encapsulan las consultas SQL y la lógica de manipulación de datos, proporcionando una interfaz limpia y orientada a objetos para acceder a la base de datos.



Al utilizar el lenguaje de programación Java junto con el patrón DAO y MySQL, se puede lograr una arquitectura de aplicaciones organizada y eficiente. El patrón DAO separa la lógica de acceso a datos de la lógica de negocio, mientras que Java y JDBC se encargan de establecer la conexión y ejecutar consultas en la base de datos MySQL. Esto permite un desarrollo modular, mantenible y escalable, donde las operaciones de base de datos están abstractas y encapsuladas en las clases DAO.

II. Interfaz de usuario

En esta etapa del proyecto, nos centraremos en la implementación de la capa de interfaz de usuario del prototipo. El objetivo es desarrollar una interfaz intuitiva y funcional que permita a los usuarios ingresar los datos del cliente y los requerimientos generales y específicos de manera efectiva.

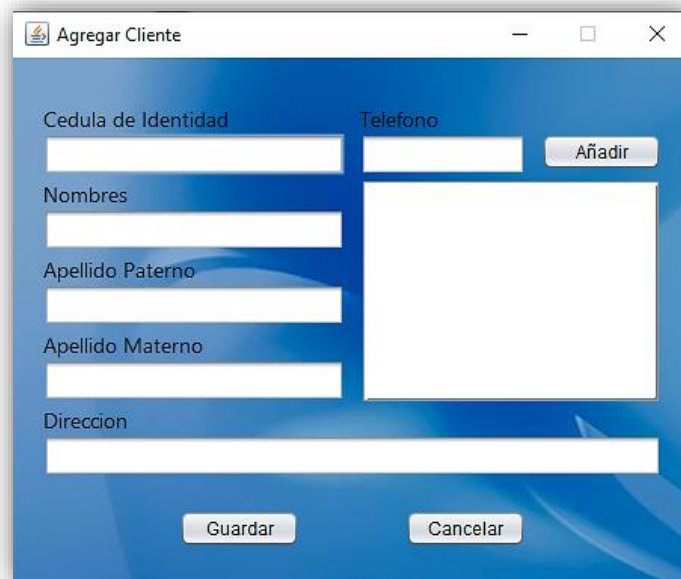
La implementación de la interfaz siguió el patrón MVC (Model-View-Controller) que proporciona una estructura organizativa para el desarrollo de aplicaciones, donde el modelo se encarga de los datos y la lógica de negocio, la vista se encarga de la interfaz de usuario y el controlador se encarga de la interacción entre el modelo y la vista. Esta separación de responsabilidades promueve el modularidad y la mantenibilidad del código.

Diseño de la interfaz

Comenzaremos por diseñar la interfaz gráfica que contendrá los elementos necesarios para ingresar los datos del cliente y los requerimientos. Esto implica definir los campos de entrada de texto, listas desplegables u otros elementos interactivos necesarios para capturar la información requerida.

Registro del cliente

En el modelo se muestran los datos o tributos de los clientes a registrar, los cuales son su cédula de identidad, nombre y teléfonos.



Se dio especial atención a la parte de los teléfonos de los clientes ya que un cliente puede tener varios teléfonos, esto se manejó utilizando una lista en la cual se puede ingresar los números de teléfono que el cliente quiera. Después, con estos datos se creará una instancia para el nuevo cliente que después será insertado en la base de datos.

Requerimientos Generales y Específicos

Ingresar Requerimientos

Requerimientos Generales

Ubicación:

Superficie:

☒ Casa ☐ Departamento ☐ Garzonier ☐ Lote

Cliente:

Tipo de Oferta:

Precio Mínimo:

Precio Máximo:

Requerimientos Específicos

Requerimientos especiales del Inmueble:

Buscar Cancelar

Cabe mencionar que se debe ingresar el CI del cliente, esto para identificar qué cliente realizó una búsqueda de un inmueble en específico para datos históricos.

Para la parte de Requerimientos generales y específicos se evidencio que no todos los inmuebles cumplen con algunos de los requerimientos específicos que pudimos identificar, es por eso que se decidió dividir los requerimientos específicos por los diferentes tipos de inmuebles que la Inmobiliaria DEGA ofrece, que son casas, departamentos, garzoniers y lotes, creando así requerimientos específicos para cada uno de ellos. A continuación, se muestra los requerimientos para cada inmueble.

☒ Casa ☐ Departamento ☐ Garzonier ☐ Lote

Requerimientos Específicos

Casa

Numero de Plantas:

Antigüedad de Construcción:

Superficie de Construcción:

Dormitorios:

Baños:

☐ Cocina ☐ Comedor ☐ Sala ☐ Patio ☐ Garaje

Requerimientos Específicos de Casas

☐ Casa ☒ Departamento ☐ Garzonier ☐ Lote

Requerimientos Específicos

Departamento

Numero de Piso:

Antigüedad de Construcción:

Numero de Estacionamiento:

Baños:

Dormitorios:

☐ Cocina ☐ Comedor ☐ Sala ☐ Permiso Mascota

Requerimientos Específicos de Departamentos

☐ Casa ☐ Departamento ☒ Garzonier ☐ Lote

Requerimientos Específicos

Garzonier

Antigüedad de Construcción:

Capacidad Máxima:

Dormitorios:

Baños:

☐ Cocina ☐ Comedor ☐ Sala ☐ Amueblado

Requerimientos Específicos de Garzoniers

☐ Casa ☐ Departamento ☐ Garzonier ☒ Lote

Requerimientos Específicos

Lote

Uso Actual:

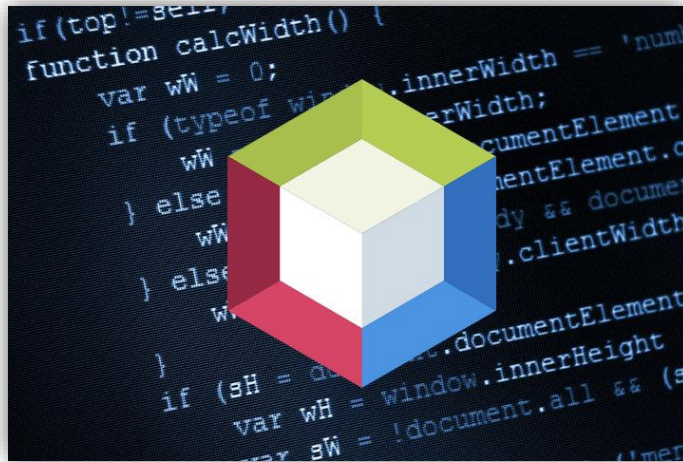
☐ Topografía

Requerimientos Específicos de Lotes

Desarrollo de la interfaz

Utilizando las herramientas de desarrollo adecuadas, implementaremos la interfaz diseñada. Esto implica la creación de los componentes gráficos, la disposición adecuada de los elementos y la asignación de eventos o acciones a cada elemento de la interfaz.

Como herramienta de trabajo, se usó Apache NetBeans 12, que es un IDE de desarrollo que proporciona una amplia gama de características y soporte para múltiples lenguajes de programación, entre los cuales se encuentra el seleccionado para el proyecto. Es conocido por su facilidad de uso, su capacidad para trabajar con diferentes tecnologías y su enfoque en la productividad del desarrollador.



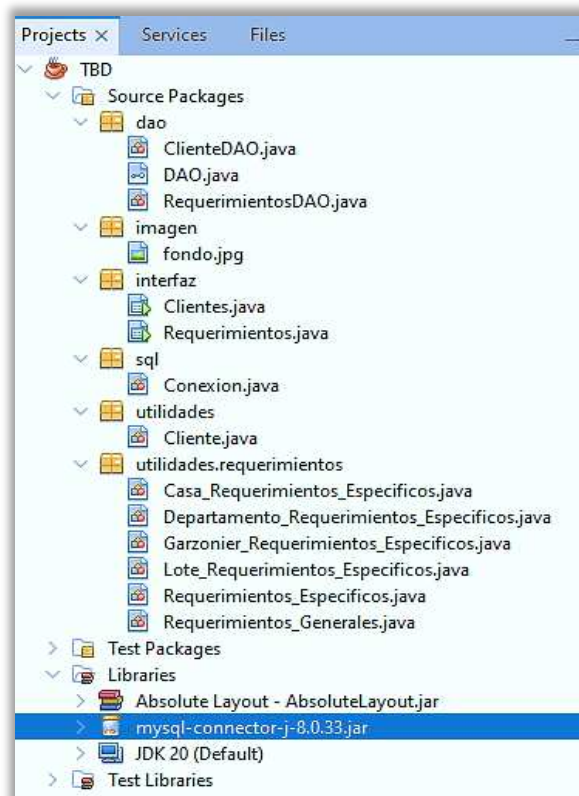
Validación de datos

A medida que los usuarios ingresan los datos, es importante implementar validaciones para garantizar la integridad de la información. Esto incluye la verificación de formatos correctos (como los nombres y apellidos de los clientes o sus números de teléfono), la validación de rangos (para el caso del número de dormitorios y baños) y la verificación de datos obligatorios y opcionales.

Captura de datos

Una vez que los usuarios ingresen los datos en la interfaz, implementaremos la lógica necesaria para capturar y almacenar esta información. Esto implica el uso de variables y estructuras de datos adecuadas para almacenar la información ingresada temporalmente.

Para el ingreso de los datos de clientes se creó una clase que representa a los clientes como objetos con los atributos que la base de datos actual tenga definidos. Tanto para los Requerimientos generales y específicos utilizamos la misma idea creando así objetos de tipo requerimientos generales y específicos.



- ❖ En el paquete “dao” se implementaron las clases “DAO”, “ClienteDAO”, “RequerimientosDAO” estas son usadas para que haga de intermediario con la base de datos por temas de requerimientos del trabajo solo se implementó el método “insertar ()” esta es implementada por las clases “ClienteDAO” y “RequerimientosDAO” el método sirve para guardar los datos en la BD, para esto se requiere una conexión con la BD que se explicara más a detalle adelante.
- ❖ El paquete “imagen” solo guarda el fondo de pantalla de los prototipos o Frames.
- ❖ El paquete “interfaz” contiene los Frames del proyecto “Clientes”, “Requerimientos” y sus respectivos diseños fueron presentados antes. Otro funcionamiento que tienen estas clases, sería que ambas obtienen los datos necesarios para la creación de los objetos tipo “Cliente” y todas las del paquete “utilidades.requerimientos” también es importante mencionar los atributos que poseen estas clases.

Por ejemplo, para la “Requerimientos” se tienen los atributos “Conexión” y “RequerimientosDAO” estos son usados para establecer una conexión con la BD (Conexion) y guardar los datos (RequerimientosDAO) para clientes es prácticamente lo mismo solo varía el objeto DAO

```
public class Requerimientos extends javax.swing.JFrame {

    private Requerimientos_Generales rg;
    private final Conexion con;
    private RequerimientosDAO rdao;
```


Interacción con la capa de DAO

Para persistir los datos ingresados en la base de datos MySQL, estableceremos una interacción entre la capa de interfaz de usuario y la capa de acceso a datos (DAO). Utilizaremos las clases y métodos DAO correspondientes para insertar los datos en las tablas de la base de datos. Se creó una interfaz para que sea implementada por las clases DAO, por ejemplo, la clase "ClienteDAO" implementa la interfaz "DAO<Cliente>"

```
package dao;

import utilidades.Cliente;
import java.sql.*;

public class ClienteDAO implements DAO<Cliente> {
    private final Connection conexion;

    public ClienteDAO(Connection conexion) {
        this.conexion = conexion;
    }

    @Override
    public void insertar(Cliente cliente) {
        try {
            conexion.setAutoCommit(false);
            String sql = "INSERT INTO clientes (ci, nombres, apellidos, direccion) VALUES (?, ?, ?, ?)";
            PreparedStatement statement = conexion.prepareStatement(sql);
            statement.setInt(1, cliente.ci);
            statement.setString(2, cliente.nombres);
            statement.setString(3, cliente.apellidos);
            statement.setString(4, cliente.direccion);
            statement.executeUpdate();
            for (String telefono : cliente.telefonos) {
                sql = "INSERT INTO telefonos (ci, numero) VALUES (?, ?)";
                statement = conexion.prepareStatement(sql);
                statement.setInt(1, cliente.ci);
                statement.setString(2, telefono);
                statement.executeUpdate();
            }
            statement.close();
            conexion.commit();
            System.out.println("Cliente insertado correctamente");
            conexion.setAutoCommit(true);
        } catch (SQLException e) {
            try {
                conexion.rollback();
                System.out.println("Error al insertar el cliente");
            } catch (SQLException ex) {
                System.out.println(ex.toString());
            }
            System.out.println(e.toString());
        }
    }
}
```

Al implementar estos pasos, lograremos una capa de interfaz de usuario funcional que permitirá a los usuarios ingresar los datos del cliente y los requerimientos generales y específicos de manera efectiva. Aseguraremos la integridad de los datos mediante validaciones adecuadas y estableceremos la conexión necesaria con la capa de DAO para almacenar los datos en la base de datos MySQL.

Ajustes en la Base de Datos

Para la implementación de los requerimientos específicos que sigue el patrón DAO se tuvo que realizar modificaciones en la estructura de la base de datos, dicho cambios no representan ningún riesgo en la integridad del estado de la base de datos actual, ya que solo se agregarán entidades para cada una de los objetos DAO que representan los requerimientos específicos de los diferentes inmuebles. Las tablas añadidas son:

```
CREATE TABLE requerimientos_especificos (  
    id_re INT PRIMARY KEY AUTO_INCREMENT,  
    id_rg INT UNIQUE,  
    sala BOOLEAN,  
    comedor BOOLEAN,  
    cocina BOOLEAN,  
    dormitorios INT,  
    banios INT,  
    antiguedad_construccion INT,  
    FOREIGN KEY (id_rg) REFERENCES requerimientos_generales (id_rg)  
);
```

```
CREATE TABLE casa_requerimientos_especificos (  
    id_cre INT PRIMARY KEY AUTO_INCREMENT,  
    id_re INT UNIQUE,  
    garaje BOOLEAN,  
    patio BOOLEAN,  
    superficie_construccion INT,  
    numero_plantas INT,  
    FOREIGN KEY (id_re) REFERENCES requerimientos_especificos (id_re)  
);
```

```
CREATE TABLE departamento_requerimientos_especificos (  
    id_dre INT PRIMARY KEY AUTO_INCREMENT,  
    id_re INT UNIQUE,  
    numero_estacionamiento VARCHAR(10),  
    permiso_mascota BOOLEAN,  
    numero_piso INT,  
    FOREIGN KEY (id_re) REFERENCES requerimientos_especificos (id_re)  
);
```

```
CREATE TABLE garzonier_requerimientos_especificos (  
    id_gre INT PRIMARY KEY AUTO_INCREMENT,  
    id_re INT UNIQUE,  
    amueblado BOOLEAN,  
    capacidad_maxima INT,  
    FOREIGN KEY (id_re) REFERENCES requerimientos_especificos (id_re)  
);
```

```
CREATE TABLE lote_requerimientos_especificos (  
    id_lre INT PRIMARY KEY AUTO_INCREMENT,  
    id_rg INT UNIQUE,  
    uso_actual VARCHAR(20),  
    topografia BOOLEAN,  
    FOREIGN KEY (id_rg) REFERENCES requerimientos_generales (id_rg)  
);
```

III. Conexión a la base de datos

En esta subsección, se describirá el proceso de conexión entre la aplicación y la base de datos utilizando la tecnología JDBC en conjunto con MySQL. A continuación, se detallarán los pasos necesarios para configurar la conexión y realizar la inserción de los datos ingresados por el usuario en la base de datos:

Controlador JDBC

En primer lugar, es necesario obtener el controlador JDBC específico para MySQL. Para este inciso se debe mencionar la librería presente en el proyecto “mysql-connector-j-8.0.33-jar” que es la tecnología JDBC, la cual se tuvo que incluir en el paquete correspondiente a las librerías.

Este paso es necesario para que la aplicación pueda utilizar el controlador JDBC y establecer la conexión con MySQL.

Establecer la conexión

Después de cargar el controlador JDBC, se establece la conexión a la base de datos MySQL utilizando la clase Connection.

```
package sql;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Conexion {

    private static final String CONTROLADOR = "com.mysql.cj.jdbc.Driver";
    private static final String URL = "jdbc:mysql://localhost:3306/inmobiliaria_DEGA";
    private static final String USUARIO = "root";
    private static final String CONTRASENIA = "12qwaszx";

    public Connection conexion;

    public Conexion() {
        try {
            Class.forName(className: CONTROLADOR);
        } catch (ClassNotFoundException e) {
            System.out.println("Error al cargar el controlador");
            System.out.println("e: " + e.toString());
        }
    }

    public void conectar() throws SQLException {
        if (conexion == null || conexion.isClosed()) {
            conexion = DriverManager.getConnection(url: URL, user: USUARIO, password: CONTRASENIA);
            System.out.println("Conexion establecida");
        }
    }

    public void desconectar() throws SQLException {
        if (conexion != null && !conexion.isClosed()) {
            conexion.close();
            System.out.println("Conexion cerrada");
        }
    }
}
```


La clase "Conexion", ubicada en el paquete "sql" del proyecto, establece la conexión a "localhost" a través del puerto "3306", que es el por defecto para MySQL. El controlador actúa como una capa intermedia entre la aplicación y la base de datos cargando el controlador específico de la base de datos.

Preparar la consulta SQL

Antes de insertar los datos en la base de datos, es necesario preparar la consulta SQL correspondiente. Puede utilizar una declaración SQL parametrizada para evitar problemas de seguridad y garantizar la integridad de los datos ingresados.

Ejecutar la consulta

Después de preparar la consulta SQL, puede ejecutarla utilizando un objeto de tipo PreparedStatement y proporcionando los valores de los parámetros correspondientes.

Cerrar la conexión

Una vez que se haya completado la inserción de datos, asegúrese de cerrar la conexión a la base de datos para liberar recursos.

Desarrollo

Al seguir estos pasos, podremos establecer la conexión entre la aplicación y la base de datos MySQL utilizando JDBC. Se mostrarán algunas capturas del código siguiendo estos pasos e indicaremos la parte del código correspondiente según corresponda. Las indicaciones seguirán el siguiente patrón de colores: Preparar la consulta SQL: **Rojo**, Ejecutar la consulta: **Amarillo** y Cerrar la conexión: **Verde**.

```
@Override
public void insertar(Cliente cliente) {
    try {
        conexion.setAutoCommit(false);
        String sql = "INSERT INTO clientes (ci, nombres, apellidos, direccion) VALUES (?, ?, ?, ?)";
        PreparedStatement statement = conexion.prepareStatement(sql);
        statement.setInt(1, cliente.ci);
        statement.setString(2, cliente.nombres);
        statement.setString(3, cliente.apellidos);
        statement.setString(4, cliente.direccion);
        statement.executeUpdate();
        for (String telefono : cliente.telefonos) {
            sql = "INSERT INTO telefonos (ci, numero) VALUES (?, ?)";
            statement = conexion.prepareStatement(sql);
            statement.setInt(1, cliente.ci);
            statement.setString(2, telefono);
            statement.executeUpdate();
        }
        statement.close();
        conexion.commit();
        System.out.println("Cliente insertado correctamente");
        conexion.setAutoCommit(true);
    } catch (SQLException e) {
        try {
            conexion.rollback();
            System.out.println("Error al insertar el cliente");
        } catch (SQLException ex) {
            System.out.println(ex.toString());
        }
        System.out.println(e.toString());
    }
}
```

```

private void insertarG(Requerimientos_Generales rg) {
    try {
        String sql = "INSERT INTO requerimientos_generales "
            + "(ci, fecha_creacion, ubicacion, tipo_inmueble, tipo_oferta, "
            + "precio_minimo, precio_maximo, superficie)"
            + " VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

        PreparedStatement statement = conexion.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
        Date fechaActual = new Date();
        statement.setInt(1, rg.ci);
        statement.setDate(2, new java.sql.Date(date: fechaActual.getTime()));
        statement.setString(3, rg.ubicacion);
        statement.setString(4, rg.tipo_inmueble);
        statement.setString(5, rg.tipo_oferta);
        statement.setInt(6, rg.precio_minimo);
        statement.setInt(7, rg.precio_maximo);
        statement.setInt(8, rg.superficie);
        statement.executeUpdate();
        ResultSet generatedKeys = statement.getGeneratedKeys();
        if (generatedKeys.next()) {
            id_rg = generatedKeys.getInt(1);
        }

        // Cerrar los recursos
        generatedKeys.close();
        statement.close();
        conexion.commit();
        System.out.println("Requerimiento general insertado correctamente");
    } catch (SQLException e) {
        try {
            conexion.rollback();
            System.out.println("Error al insertar el requerimiento general");
        } catch (SQLException ex) {
            System.out.println(ex.toString());
        }
    }
}

```

```

private void insertarC(Casa_Requerimientos_Especificos cre) {
    insertarG((Requerimientos_Generales) ((Requerimientos_Especificos) cre));
    insertarE((Requerimientos_Especificos) cre);
    try {
        String sql = "INSERT INTO casa_requerimientos_especificos "
            + "(id_re, garaje, patio, superficie_construccion, numero_plantas) "
            + "VALUES (?, ?, ?, ?, ?)";

        PreparedStatement statement = conexion.prepareStatement(sql);
        statement.setInt(1, id_re);
        statement.setBoolean(2, cre.garaje);
        statement.setBoolean(3, cre.patio);
        statement.setInt(4, cre.superficie_construccion);
        statement.setInt(5, cre.numero_plantas);
        statement.executeUpdate();

        statement.close();
        conexion.commit();
        System.out.println("Requerimiento especifico de casa insertado correctamente");
    } catch (SQLException e) {
        try {
            conexion.rollback();
            System.out.println("Error al insertar el requerimiento especifico de casa");
        } catch (SQLException ex) {
            System.out.println(ex.toString());
        }
    }
}

```

IV. Uso de estructuras e instrucciones de PL/SQL

El uso de estructuras e instrucciones de PL/SQL en la base de datos proporciona una serie de ventajas significativas para realizar operaciones de manera eficiente y garantizar la integridad de los datos ingresados. PL/SQL (Procedural Language/Structured Query Language) es un lenguaje de programación procedimental específico del Sistema de Gestión de Bases de Datos Oracle.

Una de las ventajas clave de utilizar PL/SQL es su capacidad para trabajar directamente en la base de datos, lo que minimiza la necesidad de transferir grandes volúmenes de datos entre la aplicación y la base de datos. Esto resulta en un rendimiento mejorado y una reducción en la latencia de red.

Implementación de Transacciones

Se requiere implementar transacciones en el proceso de introducción de datos, lo cual brinda la posibilidad de agrupar una serie de operaciones SQL en una unidad lógica y coherente.

Para lograr esto, se ha decidido implementar las transacciones dentro de las clases DAO. En el método "insertar (Cliente cliente)", se incluyen instrucciones clave para la implementación de transacciones. La instrucción "conexion.setAutoCommit(false)" desactiva la realización automática de *commits* en la base de datos, lo que permite controlar manualmente el momento en que se confirman los cambios. Luego, la instrucción "conexion.commit()" se utiliza para realizar el *commit* manual de las operaciones si todas las inserciones se ejecutan correctamente. En caso contrario, se utiliza el comando "conexion.rollback()" para deshacer los cambios y volver al estado anterior. Por último, la instrucción "conexion.setAutoCommit(true)" se emplea para reactivar la realización automática de *commits*.

```
@Override
public void insertar(Cliente cliente) {
    try {
        conexion.setAutoCommit(false);
        String sql = "INSERT INTO clientes (ci, nombres, apellidos, direccion) VALUES (?, ?, ?, ?)";
        PreparedStatement statement = conexion.prepareStatement(sql);
        statement.setInt(1, cliente.ci);
        statement.setString(2, cliente.nombres);
        statement.setString(3, cliente.apellidos);
        statement.setString(4, cliente.direccion);
        statement.executeUpdate();
        for (String telefono : cliente.telefonos) {
            sql = "INSERT INTO telefonos (ci, numero) VALUES (?, ?)";
            statement = conexion.prepareStatement(sql);
            statement.setInt(1, cliente.ci);
            statement.setString(2, telefono);
            statement.executeUpdate();
        }
        statement.close();
        conexion.commit();
        System.out.println("Cliente insertado correctamente");
        conexion.setAutoCommit(true);
    } catch (SQLException e) {
```

```

    } catch (SQLException e) {
        try {
            conexion.rollback();
            System.out.println("Error al insertar el cliente");
        } catch (SQLException ex) {
            System.out.println("ex: " + ex.toString());
        }
        System.out.println("e: " + e.toString());
    }
}

```

Mas capturas de transacciones

```

@Override
public void insertar(Object object) {
    try {
        conexion.setAutoCommit (b1n:false);
        if (object instanceof Casa_Requerimientos_Especificos) {
            insertarC((Casa_Requerimientos_Especificos) object);
        } else if (object instanceof Departamento_Requerimientos_Especificos) {
            insertarD((Departamento_Requerimientos_Especificos) object);
        } else if (object instanceof Garzonier_Requerimientos_Especificos) {
            insertarG((Garzonier_Requerimientos_Especificos) object);
        } else {
            insertarL((Lote_Requerimientos_Especificos) object);
        }
        conexion.setAutoCommit (b1n:true);
    } catch (SQLException ex) {
        System.out.println("ex: " + ex.toString());
    }
}

```

```

private void insertarG(Requerimientos_Generales rg) {
    try {
        String sql = "INSERT INTO requerimientos_generales "
            + "(ci, fecha_creacion, ubicacion, tipo_inmueble, tipo_oferta, "
            + "precio_minimo, precio_maximo, superficie)"
            + " VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement statement = conexion.prepareStatement (string: sql, i: Statement.RETURN_GENERATED_KEYS);
        // Cerrar los recursos
        generatedKeys.close();
        statement.close();
        conexion.commit();
        System.out.println("Requerimiento general insertado correctamente");
    } catch (SQLException e) {
        try {
            conexion.rollback();
            System.out.println("Error al insertar el requerimiento general");
        } catch (SQLException ex) {
            System.out.println("ex: " + ex.toString());
        }
    }
}

```

Conclusión

La implementación de transacciones en las clases DAO permite asegurar la integridad de los datos introducidos, ya que se pueden revertir los cambios en caso de algún error. Esto se logra mediante el uso de instrucciones específicas para controlar manualmente los *commits* y *rollbacks* en la base de datos.

V. Documentación y explicación de la conexión nativa

Para este prototipo, no se utilizará una conexión nativa, ya que se ha optado por utilizar JDBC en conjunto con MySQL para establecer la conexión a la base de datos. Sin embargo, es importante comprender qué es una conexión nativa y cuándo puede ser necesaria.

Ventajas de la nativa en MySQL

Una conexión nativa se refiere a una comunicación directa entre la aplicación y la base de datos, sin intermediarios adicionales como ODBC o JDBC. En el caso de MySQL, se puede utilizar una conexión nativa basada en el protocolo de MySQL, diseñado específicamente para la comunicación con esta base de datos.

La conexión nativa en MySQL aprovecha las características y funcionalidades específicas de este sistema de gestión de bases de datos, permitiendo una interacción eficiente y optimizada. Al establecer una conexión nativa en MySQL, se evita la necesidad de utilizar controladores adicionales o capas de abstracción, lo que puede resultar en una mayor simplicidad y rendimiento en el acceso a la base de datos. Sin embargo, su uso está limitado a la base de datos específica para la cual está diseñada, en este caso, MySQL.

Uso de JDBC como tecnologías de acceso a la base de datos a MySQL

En el contexto de este prototipo, se ha optado por utilizar JDBC y MySQL como tecnologías de acceso a la base de datos. JDBC es una API estándar de Java que ofrece una capa de abstracción para trabajar con diferentes bases de datos relacionales, incluyendo MySQL. Proporciona una forma portátil y eficiente de establecer conexiones, ejecutar consultas y administrar transacciones.

Ventajas de elegir JDBC y MySQL sobre una conexión nativa

La elección de JDBC y MySQL en lugar de una conexión nativa se basa en varios factores, incluyendo la familiaridad y conocimiento del equipo con Java y JDBC, la disponibilidad de controladores JDBC para MySQL y la capacidad de JDBC para proporcionar una capa de abstracción más amplia y portabilidad en caso de que se desee cambiar a otro sistema de gestión de bases de datos en el futuro.

En resumen, para este prototipo, se ha optado por utilizar JDBC en conjunto con MySQL para establecer la conexión a la base de datos. Aunque una conexión nativa ofrece ventajas en términos de rendimiento y eficiencia, la elección de JDBC se debe a su portabilidad, facilidad de uso y capacidad para trabajar con diferentes bases de datos relacionales, pero principalmente a la familiaridad del equipo de desarrollo que se tiene con Java y diferentes herramientas relacionadas con este lenguaje de programación.

Resultados de Pruebas realizadas

En esta sección se muestran los resultados de las pruebas realizadas a la aplicación, entre las pruebas que se realizaron se tiene la verificación de la conexión, el registro de un nuevo cliente y los requerimientos que los clientes tiene sobre cada uno de los inmuebles, así como algunos posibles casos de error.

Estado inicial de las tablas

Tabla "clientes"

	ci	nombres	apellidos	direccion
▶	8456789	Guillermo	Rojas Becerra	Avenida Arce 234, Sucre, Bolivia
	8965432	Daniel Boris	Limachi cuiza	Calle Murillo 789,
	8976543	Andres Juan Jose	Revollo Huanca	Plaza 14 de Septiembre #789
	9123456	Cristian	García Rodríguez	Calle Sucre #456
	9394750	Emerson Denis	Vera Viscarra	Av. Cap. Victor Ustariz km 6
*	NULL	NULL	NULL	NULL

Tabla "telefonos"

id_telefono	ci	numero
1	9394750	71234567
2	9394750	71345678
3	9394750	71456789
4	9123456	71567890
5	9123456	71678901
6	8976543	71789012
7	8976543	71890123
8	8456789	71901234
9	8456789	72012345
10	8965432	72123456
11	8965432	72234567
12	8965432	72345678
NULL	NULL	NULL

Actualmente no existen datos en las tablas de requerimientos.

Agregar un nuevo Cliente

Ahora se procederá a la ejecución del programa para demostrar la conexión existente entre la aplicación y la BD, esto se lo realizará añadiendo un nuevo cliente.

Agregar Cliente

Cedula de Identidad: 9493751

Telefono: [Empty] Añadir

Nombres: Sandra

Apellido Paterno: Utrunco

Apellido Materno: Moya

Direccion: Plaza Simon Bolivar, Quillacollo

Guardar Cancelar

```
run:
Conexion establecida
Cliente insertado correctamente
Conexion cerrada
```

Tabla “clientes”

ci	nombres	apellidos	direccion
8456789	Guillermo	Rojas Becerra	Avenida Arce 234, Sucre, Bolivia
8965432	Daniel Boris	Limachi cuiza	Calle Murillo 789,
8976543	Andres Juan Jose	Revollo Huanca	Plaza 14 de Septiembre #789
9123456	Cristian	García Rodríguez	Calle Sucre #456
9394750	Emerson Denis	Vera Viscarra	Av. Cap. Victor I Istariz km 6
9493751	Sandra	Utuonco Moya	Plaza Simon Bolivar, Quillacollo

Tabla “telefonos”

id_telefono	ci	numero
1	9394750	71234567
2	9394750	71345678
3	9394750	71456789
4	9123456	71567890
5	9123456	71678901
6	8976543	71789012
7	8976543	71890123
8	8456789	71901234
9	8456789	72012345
10	8965432	72123456
11	8965432	72234567
12	8965432	72345678
13	9493751	72218135
14	9493751	71418129

Ahora se procederá a insertar un cliente con la misma cédula de identidad.

Agregar Cliente

Cedula de Identidad

94933751

Telefono

Añadir

Nombres

Victor

76847361

77823310

4377450

Apellido Paterno

Maldonado

Apellido Materno

Gutierrez

Direccion

Calle Murrillo, La Paz

Guardar

Cancelar

```

Conexion establecida
Error al insertar el cliente
java.sql.SQLIntegrity
ConstraintViolationException:
Duplicate entry '9493751'
for key 'clientes.PRIMARY'
Conexion cerrada
  
```

El programa hizo *rollback* por ende los datos siguen iguales.

Se hará la demostración con las listas de requerimientos, para esto se utilizará los “ci” de los clientes existentes, empezando con el del cliente antes insertado.

Requerimiento de una Casa

Ingresar Requerimientos

Requerimientos Generales

Ubicacion

Superficie

☒ Casa ☐ Departamento ☐ Garzonier ☐ Lote

Cliente

Tipo de Oferta

Precio Minimo Precio Maximo

Requerimientos Especificos

Casa

Numero de Plantas

Antiguedad de Construccion

Superficie de Construccion

☒ Cocina ☐ Comedor ☒ Sala ☒ Patio ☒ Garaje

Dormitorios

Baños

Buscar

Cancelar

```
run:
Conexion establecida
Requerimiento general insertado correctamente
Requerimiento especifico insertado correctamente
Requerimiento especifico de casa insertado correctamente
Conexion cerrada
```

Tabla “requerimientos_generales”

[illegible]

Tabla “requerimientos_especificos”

[illegible]

Tabla "casa_requerimientos_especificos"

id_cre	id_re	garaje	patio	superficie_construccion	numero_plantas
1	1	1	1	5000	2
NULL	NULL	NULL	NULL	NULL	NULL

Requerimiento de un Departamento

ci	nombres	apellidos	direccion
8456789	Guillermo	Rojas Becerra	Avenida Arce 234, Sucre, Bolivia
8965432	Daniel Boris	Limachi cuiza	Calle Murillo 789,
8976543	Andres Juan Jose	Revollo Huanca	Plaza 14 de Septiembre #789
9123456	Cristian	García Rodríguez	Calle Sucre #456
9394750	Emerson Denis	Vera Viscarra	Av. Cap. Victor Ustariz km 6
9493751	Sandra	Uturonco Moya	Plaza Simon Bolivar, Quillacollo
NULL	NULL	NULL	NULL

Ingresar Requerimientos

Requerimientos Generales

Ubicacion
calle jordan y lanza

Superficie
2000

☐ Casa
☒ Departamento
☐ Garzonier
☐ Lote

Cliente
9394750

Tipo de Oferta
Venta

Precio Minimo
15000

Precio Maximo
16000

Requerimientos Especificos

Departamento

Numero de Piso
4

Antiguedad de Construccion
20

Numero de Estacionamiento
645

Baños
1

Dormitorios
1

☒ Cocina
☒ Comedor
☐ Sala
☒ Permiso Mascota

Buscar

Cancelar

Conexion establecida
Requerimiento general insertado correctamente
Requerimiento especifico insertado correctamente
Requerimiento especifico de departamento insertado correctamente
Conexion cerrada

Tabla "requerimientos_generales"

id_rg	ci	fecha_creacion	ubicacion	tipo_inmueble	tipo_oferta	precio_minimo	precio_maximo	superficie
1	9493751	2023-06-27	zona norte	Casa	Venta	10000	15000	20000
2	9394750	2023-06-27	calle jordan y lanza	Departamento	Venta	15000	16000	2000
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabla "requerimientos_especificos"

id_re	id_rg	sala	comedor	cocina	dormitorios	banios	antiguedad_construccion
1	1	1	0	1	2	2	10
2	2	0	1	1	1	1	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabla "departamento_requerimientos_especificos"

id_dre	id_re	numero_estacionamiento	permiso_mascota	numero_piso
1	2	645	1	4
NULL	NULL	NULL	NULL	NULL

Requerimiento de un Garzoniers

ci	nombres	apellidos	direccion
8456789	Guillermo	Rojas Becerra	Avenida Arce 234, Sucre, Bolivia
8965432	Daniel Boris	Limachi cuiza	Calle Murillo 789,
8976543	Andres Juan Jose	Revollo Huanca	Plaza 14 de Septiembre #789
9123456	Cristian	García Rodríguez	Calle Sucre #456
9394750	Emerson Denis	Vera Viscarra	Av. Cap. Victor Ustariz km 6
9493751	Sandra	Utuonco Moya	Plaza Simon Bolivar, Quillacollo
NULL	NULL	NULL	NULL

Ingresar Requerimientos

Requerimientos Generales

Ubicacion

Calle Avaroa #55

Superficie

5000

☐ Casa
☐ Departamento
☒ Garzonier
☐ Lote

Precio Minimo

2000

Precio Maximo

2300

Cliente

8965432

Tipo de Oferta

Alquiler

Requerimientos Especificos

Garzonier

Antiguedad de Construccion

5

Capacidad Maxima

3

☒ Cocina
☒ Comedor
☒ Sala
☒ Amueblado

Dormitorios

3

Baños

2

Buscar

Cancelar

Conexion establecida
 Requerimiento general insertado correctamente
 Requerimiento especifico insertado correctamente
 Requerimiento especifico de garzonier insertado correctamente
 Conexion cerrada

Tabla "requerimientos_generales"

id_rg	ci	fecha_creacion	ubicacion	tipo_inmueble	tipo_oferta	precio_minimo	precio_maximo	superficie
1	9493751	2023-06-27	zona norte	Casa	Venta	10000	15000	20000
2	9394750	2023-06-27	calle jordan y lanza	Departamento	Venta	15000	16000	2000
4	8965432	2023-06-27	Calle Avaroa #55	Garzonier	Alquiler	2000	2300	5000
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Página 41 de 43

Tabla "requerimientos_especificos"

id_re	id_rg	sala	comedor	cocina	dormitorios	banios	antiguedad_construccion
1	1	1	0	1	2	2	10
2	2	0	1	1	1	1	20
3	4	1	1	1	3	2	5
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabla "garzonier_requerimientos_especificos"

id_gre	id_re	amueblado	capacidad_maxima
1	3	1	3
NULL	NULL	NULL	NULL

Requerimiento de un Lote

Ingresar Requerimientos

Requerimientos Generales

Ubicacion

Av Peru y Blanco Galindo

Superficie

5500

☐ Casa
☐ Departamento
☐ Garzonier
☒ Lote

Precio Minimo

68000

Precio Maximo

70000

Cliente

8965432

Tipo de Oferta

Venta

Requerimientos Especificos

Lote

Uso Actual

Residencial

☒ Topografia

Buscar

Cancelar

Conexion establecida
 Requerimiento general insertado correctamente
 Requerimiento especifico de lote insertado correctamente
 Conexion cerrada

Tabla "requerimientos_generales"

id_rg	ci	fecha_creacion	ubicacion	tipo_inmueble	tipo_oferta	precio_minimo	precio_maximo	superficie
1	9493751	2023-06-27	zona norte	Casa	Venta	10000	15000	20000
2	9394750	2023-06-27	calle jordan y lanza	Departamento	Venta	15000	16000	2000
4	8965432	2023-06-27	Calle Avaroa #55	Garzonier	Alquiler	2000	2300	5000
5	8965432	2023-06-27	Av Peru y Blanco Galindo	Lote	Venta	68000	70000	5500
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabla "lote_requerimientos_especificos"

id_lre	id_rg	uso_actual	topografia
2	5	Residencial	1
NULL	NULL	NULL	NULL

Requerimiento con un "ci" inexistente

Finalmente se probará el uso de las transacciones para esto se ingresa un cliente no existente.

```
Error al insertar el requerimiento general
java.sql.SQLIntegrityConstraintViolationException: Cannot add or update a child row: a foreign key constraint fails
Error al insertar el requerimiento especifico de lote
java.sql.SQLIntegrityConstraintViolationException: Cannot add or update a child row: a foreign key constraint fails
Conexion cerrada
```

Tabla "requerimientos_generales"

id_rg	ci	fecha_creacion	ubicacion	tipo_inmueble	tipo_oferta	precio_minimo	precio_maximo	superficie
1	9493751	2023-06-27	zona norte	Casa	Venta	10000	15000	20000
2	9394750	2023-06-27	calle jordan y lanza	Departamento	Venta	15000	16000	2000
4	8965432	2023-06-27	Calle Avaroa #55	Garzonier	Alquiler	2000	2300	5000
5	8965432	2023-06-27	Av Peru y Blanco Galindo	Lote	Venta	68000	70000	5500
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabla "lote_requerimientos_especificos"

id_lre	id_rg	uso_actual	topografia
2	5	Residencial	1
NULL	NULL	NULL	NULL

El *rollback* conservó la integridad de los datos.