**Universidad Mariano Gálvez de Guatemala**

**Facultad de Ingeniería en Sistemas de la Información**

**Curso: Analisis de Sistemas**

**7mo. Semestre**

**PROYECTO FINAL**

**José Armando López Cruz 7690-21-2673**

**Marvin Geobany Reyna Ortega 7690-22-8291**

**Victor Alfredo Macario Enríquez 7690-22-5042**

**Pedro Javier De León Coy 7690-22-8894**

**Guatemala 22 mayo 2025**

**Descripción general**

Esta página ofrece una introducción general al sistema Deploy-Analisis-Registros, que abarca su propósito, arquitectura y componentes clave. Para obtener información detallada sobre la configuración de contenedores, consulte [Servicios de contenedores](https://deepwiki.com/Re-21-12/Deploy-Analisis-Registros/2.1-container-services). Para obtener información completa sobre el diseño de bases de datos, consulte [Diseño de bases de datos](https://deepwiki.com/Re-21-12/Deploy-Analisis-Registros/3-database-design). Para obtener instrucciones de implementación, consulte [la Guía de implementación](https://deepwiki.com/Re-21-12/Deploy-Analisis-Registros/4-deployment-guide).

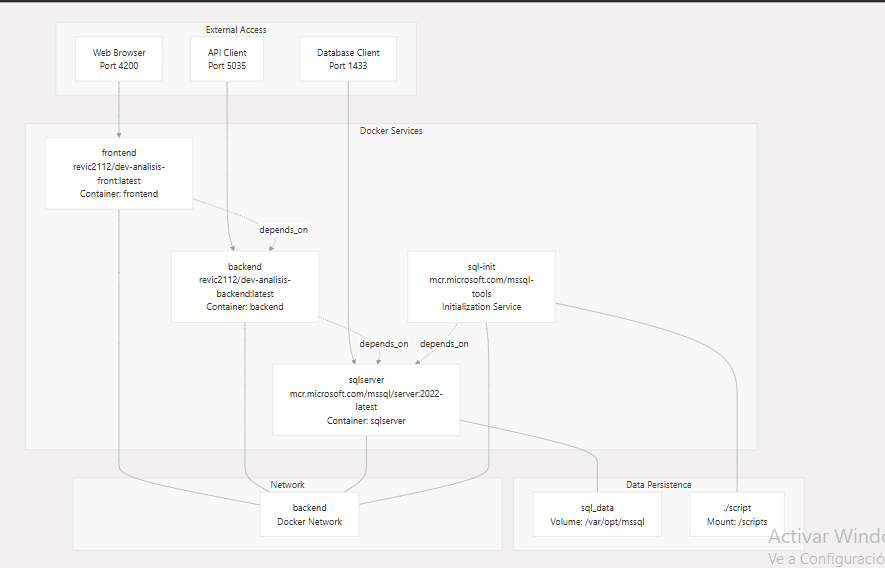
**Propósito y alcance**

Deploy-Analisis-Registros es un sistema contenedorizado de registro y análisis de personas, diseñado para gestionar registros individuales con datos geográficos y de clasificación. El sistema proporciona una plataforma web completa para almacenar, organizar y acceder a la información personal, incluyendo datos personales, asociaciones regionales, clasificaciones de tipo y seguimiento de estado.

La arquitectura del sistema consta de cuatro servicios Docker orquestados a través de Docker Compose, que proporcionan una aplicación completa de tres niveles con interfaz web, backend API y persistencia de base de datos SQL Server.

**Arquitectura del sistema**

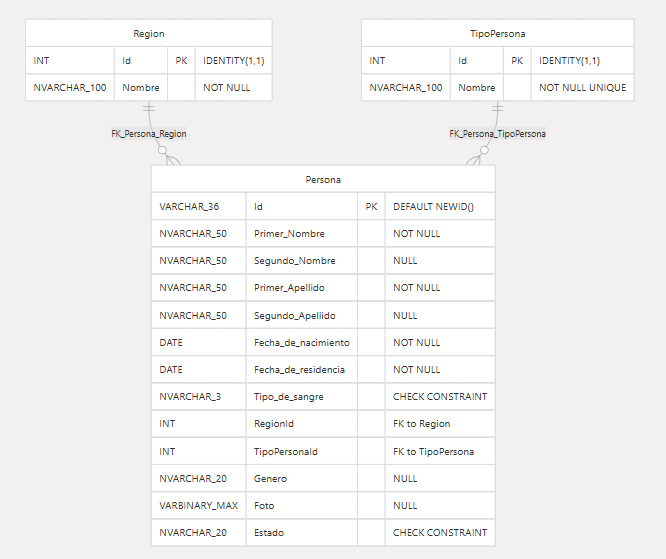
La aplicación sigue un patrón de microservicios en contenedores con los siguientes servicios principales definidos en: [docker-compose.yml3-49](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L3-L49):



Fuentes:[docker-compose.yml1-57](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L1-L57)

**Arquitectura de datos**

El sistema implementa un esquema de base de datos relacional en la **RegistroPersona**base de datos con tres entidades principales:



**Componentes principales**

| **Componente** | **Servicio Docker** | **Imagen** | **Puerto** | **Objetivo** |
| --- | --- | --- | --- | --- |
| Interfaz web | **frontend** | **revic2112/dev-analisis-front:latest** | 4200:80 | Interfaz de usuario para el registro de personas |
| API Backend | **backend** | **revic2112/dev-analisis-backend:latest** | 5035:5035 | Servicio de API REST |
| Base de datos | **sqlserver** | **mcr.microsoft.com/mssql/server:2022-latest** | 1433:1433 | SQL Server 2022 Express |
| Inicialización de la base de datos | **sql-init** | **mcr.microsoft.com/mssql-tools** | - | Configuración del esquema y los datos semilla |

Fuentes:[docker-compose.yml4-49](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L4-L49)

**Descripción general del esquema de base de datos**

La **RegistroPersona**base de datos contiene tres tablas principales:

* **Region**- Regiones geográficas (Norte, Sur, Este, Oeste, Centro)
* **TipoPersona**- Clasificaciones de personas (Agente, Civil, Administrador)
* **Persona**- Registros individuales con datos personales, relaciones de clave externa a Región y TipoPersona

El esquema incluye validación de datos a través de restricciones de verificación para tipos de sangre (CHK**\_TipoSangre**) y valores de estado ( **CHK\_Estado**), además de un índice optimizado **IX\_Persona\_Nombre**para consultas basadas en nombres.

Fuentes: [script/init.sql25-103](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/script/init.sql#L25-L103)

**Puntos de acceso**

* **Interfaz web**[: **http://localhost:4200**- Aplicación frontend](: http:/localhost:4200-%20Aplicación%20frontend)
* **Punto final de API**: **http://localhost:5035**API REST de backend
* **Base de datos**: **localhost:1433**- SQL Server (usuario SA, contraseña: **YourStrong!Passw0rd**)

**Inicialización del sistema**

El sistema utiliza un proceso de inicialización de varias etapas donde **sql-init** se ejecuta el servicio. [script/init.sql1-160](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/script/init.sql#L1-L160) Una vez que SQL Server esté disponible, este script crea la **RegistroPersona**base de datos, define el esquema de la tabla, establece restricciones y relaciones, e inserta los datos iniciales, incluyendo regiones predefinidas y registros de usuario de muestra.

Fuentes: [docker-compose.yml41-49](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L41-L49) [script/init.sql1-160](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/script/init.sql#L1-L160)

**Arquitectura del sistema**

Este documento describe la arquitectura general del sistema de registro de personas Deploy-Analisis-Registros, incluyendo su composición de servicios en contenedores, las relaciones de dependencia y la estructura de implementación. El sistema implementa un patrón de aplicación en contenedores de tres niveles utilizando Docker Compose para la orquestación.

**Descripción arquitectónica**

El sistema Deploy-Analisis-Registros implementa una **arquitectura contenedorizada de tres niveles** diseñada para la gestión y el análisis de registros de personas. El sistema consta de cuatro servicios Docker orquestados mediante Docker Compose, lo que proporciona una pila completa de aplicaciones web.

**Composición y dependencias del servicio**

El sistema está compuesto por cuatro servicios Docker interconectados con relaciones de dependencia explícitas que garantizan la secuencia de inicio adecuada y la disponibilidad del servicio.

La **depends\_on**configuración asegura la siguiente secuencia de inicio:

1. **El contenedor sqlserver** inicia e inicializa SQL Server 2022
2. **Los servicios backend** y **sql-init** esperan la disponibilidad de sqlserver
3. **El servicio frontend** espera la preparación del backend
4. **sql-init** ejecuta la inicialización de la base de datos una vez y sale

Fuentes: [docker-compose.yml10-11](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L10-L11) [docker-compose.yml21-22](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L21-L22) [docker-compose.yml43-44](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L43-L44).

**Arquitectura de red y comunicación**

Todos los servicios se comunican a través de una red de puente Docker dedicada denominada **backend**, que proporciona una red interna aislada con capacidades de descubrimiento de servicios.

| **Servicio** | **Puerto externo** | **Puerto interno** | **Protocolo** | **Objetivo** |
| --- | --- | --- | --- | --- |
| Interfaz | 4200 | 80 | HTTP | Acceso a la interfaz web |
| backend | 5035 | 5035 | HTTP/REST | Acceso al punto final de la API |
| servidor SQL | 1433 | 1433 | SQL/TCP | Conexiones de base de datos |
| sql-init | Ninguno | Ninguno | Interno | Sólo inicialización |

Fuentes:[docker-compose.yml12-13](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L12-L13), [docker-compose.yml23-24](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L23-L24), [docker-compose.yml38-39](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L38-L39), [docker-compose.yml48-49](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L48-L49), [docker-compose.yml55-56](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L55-L56).

**Configuración de almacenamiento y persistencia**

El sistema implementa almacenamiento de datos persistente a través de volúmenes Docker y montajes de enlaces para garantizar la durabilidad de los datos y el acceso al script de inicialización.

| **Tipo de almacenamiento** | **Fuente** | **Objetivo** | **Objetivo** |
| --- | --- | --- | --- |
| Volumen nombrado | datos\_sql | /var/opt/mssql | Persistencia de datos de SQL Server |
| Montura de enlace | ./guion | /scripts | Scripts de inicialización de bases de datos |

El **sql-init**servicio utiliza un montaje de enlace para acceder al script de inicialización desde el sistema de archivos del host, mientras que el **sqlserver**servicio utiliza un volumen con nombre para la persistencia de datos en los reinicios del contenedor.

Fuentes: [docker-compose.yml36-37](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L36-L37), [docker-compose.yml45-46](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L45-L46), [docker-compose.yml51-52](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L51-L52)

**Arquitectura de integración de bases de datos**

El sistema se integra con una base de datos SQL Server 2022 a través de un proceso de inicialización de dos fases que crea el **RegistroPersona** esquema de la base de datos y lo llena con datos semilla.

El proceso de inicialización crea un esquema de base de datos relacional completo con:

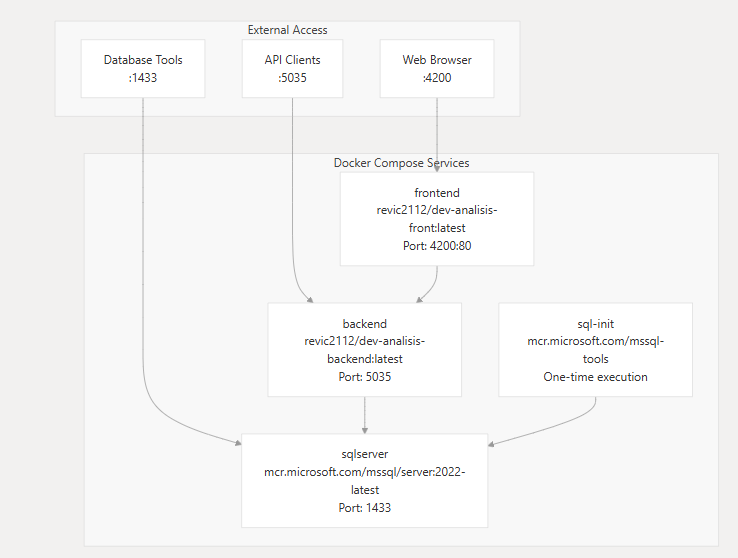
* **Tabla de regiones** para clasificación geográfica
* Tabla **TipoPersona para la categorización de tipos de personas**
* **Tabla Persona** con relaciones de clave externa a Región y TipoPersona
* Verifique las restricciones para la validación del tipo de sangre ( **CHK\_TipoSangre**) y la validación del estado ( **CHK\_Estado**)
* Optimización del rendimiento a través del **IX\_Persona\_Nombre**

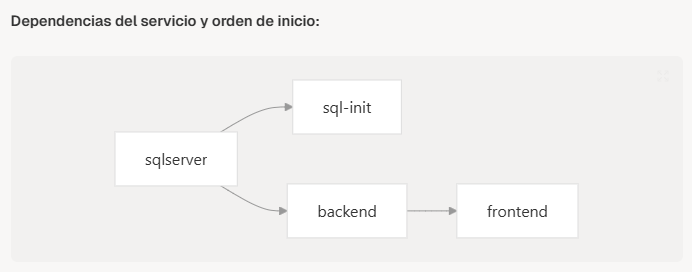
**Servicios de contenedores**

Este documento detalla los cuatro servicios de Docker definidos en el docker-compose.ymlarchivo de configuración que constituyen el sistema de registro de personas Deploy-Analisis-Registros. Cada servicio cumple una función específica en la arquitectura contenedorizada de tres niveles, desde la persistencia de datos hasta la entrega de la interfaz de usuario.

**Descripción general del servicio**

El sistema consta de cuatro servicios Docker distintos administrados a través de la orquestación Docker Compose:





**Servicio de backend**

El **backend** servicio proporciona la capa de API utilizando una imagen Docker prediseñada alojada en Docker Hub.

| **Configuración** | **Valor** |
| --- | --- |
| **Imagen** | **revic2112/dev-analisis-backend:latest** |
| **Nombre del contenedor** | **backend** |
| **Mapeo de puertos** | **5035:5035** |
| **Política de reinicio** | **always** |
| **Red** | **backend** |
| **Dependencias** | **sqlserver** |

La configuración del servicio backend incluye funciones de reinicio automático y depende de su **sqlserver** disponibilidad antes del inicio. El servicio expone el puerto 5035 para la comunicación mediante la API REST.

**Servicio de interfaz**

El **frontend**servicio entrega la interfaz de la aplicación web utilizando una imagen Docker basada en Angular.

| **Configuración** | **Valor** |
| --- | --- |
| **Imagen** | **revic2112/dev-analisis-front:latest** |
| **Nombre del contenedor** | **frontend** |
| **Mapeo de puertos** | **4200:80** |
| **Política de reinicio** | **always** |
| **Red** | **backend** |
| **Dependencias** | **backend** |

El frontend asigna el puerto interno del contenedor 80 al puerto externo 4200, lo que proporciona acceso web a la interfaz de registro de personas. El servicio depende del **backend** servicio para la conectividad de la API.

**Servicio de SQL Server**

El **sqlserver**servicio proporciona el motor de base de datos utilizando Microsoft SQL Server 2022 Express Edition.

| **Configuración** | **Valor** |
| --- | --- |
| **Imagen** | **mcr.microsoft.com/mssql/server:2022-latest** |
| **Nombre del contenedor** | **sqlserver** |
| **Mapeo de puertos** | **1433:1433** |
| **Política de reinicio** | **always** |
| **Red** | **backend** |
| **Montaje de volumen** | **sql\_data:/var/opt/mssql** |

**Variables de entorno**

| **Variable** | **Valor** | **Objetivo** |
| --- | --- | --- |
| **ACCEPT\_EULA** | **Y** | Acepta el acuerdo de licencia de SQL Server |
| **SA\_PASSWORD** | **YourStrong!Passw0rd** | Contraseña del administrador del sistema |
| **MSSQL\_PID** | **Express** | Especificación de la edición de SQL Server |

El servicio incluye almacenamiento persistente a través del **sql\_data**volumen montado **/var/opt/mssql**para preservar los archivos de base de datos luego de los reinicios del contenedor.

**Servicio de inicialización de SQL**

El **sql-init**servicio ejecuta scripts de inicialización de base de datos utilizando herramientas de línea de comandos de SQL Server.

| **Configuración** | **Valor** |
| --- | --- |
| **Imagen** | **mcr.microsoft.com/mssql-tools** |
| **Dependencias** | **sqlserver** |
| **Red** | **backend** |
| **Montaje de volumen** | **./script:/scripts** |
| **Punto de entrada** | Comando bash personalizado para ejecución de SQL |

**Proceso de inicialización**

El servicio implementa una secuencia de inicialización específica:

1. **Retraso de inicio**: suspensión de 15 segundos para garantizar la disponibilidad de SQL Server
2. **Ejecución de script**: se ejecuta **/scripts/init.sql**usando**sqlcmd**
3. **Autenticación**: Se conecta como **sa**usuario con contraseña configurada
4. **Servidor de destino**: se conecta al **sqlserver**contenedor por nombre

/bin/bash -c "sleep 15; /opt/mssql-tools/bin/sqlcmd -S sqlserver -U sa -P YourStrong!Passw0rd -i /scripts/init.sql"

**Configuración de volumen y red**

**Volúmenes con nombre**

| **Nombre del volumen** | **Objetivo** | **Punto de montaje** |
| --- | --- | --- |
| **sql\_data** | Persistencia de la base de datos de SQL Server | **/var/opt/mssql** |

**Redes**

| **Nombre de la red** | **Tipo** | **Servicios** |
| --- | --- | --- |
| **backend** | Puente | Los cuatro servicios |

**Monturas de enlace**

| **Ruta del host** | **Ruta del contenedor** | **Servicio** | **Objetivo** |
| --- | --- | --- | --- |
| **./script** | **/scripts** | **sql-init** | Scripts de inicialización |

La **backend**red permite la comunicación entre servicios utilizando nombres de contenedores como nombres de host, lo que permite que el **backend**servicio se conecte a la base de datos **sqlserver**y **sql-init**ejecute scripts en ella.

**Dependencias de servicios y orquestación**

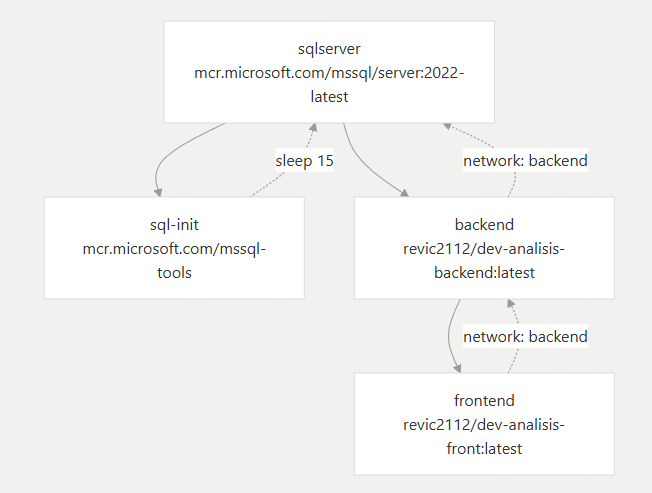
Este documento explica cómo se organizan los cuatro servicios de Docker en el sistema de registro de personas y cómo dependen entre sí durante el inicio y la ejecución. Abarca la cadena de dependencias definida en el archivo de composición, los mecanismos de sincronización del inicio y los patrones de comunicación entre servicios.

Para obtener información sobre las configuraciones individuales de los contenedores, consulte [Servicios de contenedores](https://deepwiki.com/Re-21-12/Deploy-Analisis-Registros/2.1-container-services) . Para obtener información sobre redes y almacenamiento, consulte [Redes y almacenamiento](https://deepwiki.com/Re-21-12/Deploy-Analisis-Registros/2.3-networking-and-storage) .

**Cadena de dependencia del servicio**

El sistema define una jerarquía de dependencia clara que garantiza que los servicios se inicien en el orden correcto para evitar fallas de conexión y errores de inicialización.

**Gráfico de dependencia**



Las **depends\_on**directivas crean la siguiente cadena de dependencia:

* **sql-init**depende de**sqlserver**
* **backend**depende de**sqlserver**
* **frontend**depende de**backend**

**Secuencia y tiempo de inicio**

La orquestación implementa mecanismos de sincronización específicos para gestionar los retrasos en la inicialización del servicio y garantizar el orden de inicio adecuado.

**Mecanismos de sincronización**

| **Servicio** | **Estrategia de tiempo** | **Implementación** |
| --- | --- | --- |
| **sqlserver** | Inicio inmediato | Servicio base sin dependencias |
| **sql-init** | Ejecución retrasada | **sleep 15**antes de ejecutar sqlcmd |
| **backend** | Espera de dependencia | **depends\_on: sqlserver** |
| **frontend** | Espera de dependencia | **depends\_on: backend** |

El **sql-init**servicio utiliza un mecanismo de retraso de 15 segundos: **/bin/bash -c "sleep 15; /opt/mssql-tools/bin/sqlcmd..."**para garantizar que SQL Server esté completamente listo antes de intentar la inicialización de la base de datos.

**Comunicación entre servicios**

Todos los servicios se comunican a través de la **backend**red, lo que permite el descubrimiento de servicios por nombre de contenedor y la comunicación interna segura.

**Descubrimiento de servicios**

Los servicios utilizan nombres de contenedores para la comunicación interna:

* **sql-init**se conecta a SQL Server mediante el nombre de host**sqlserver**
* **backend**El servicio se conecta a la base de datos usando el nombre de host**sqlserver**
* **frontend**El servicio se conecta a la API mediante el nombre de host**backend**

El comando sqlcmd **sql-init**demuestra este patrón: **-S sqlserver**utiliza el nombre del contenedor para el descubrimiento de servicios.

**Comportamiento de reinicio del contenedor**

Todos los servicios están configurados con **restart: always**políticas que garantizan la recuperación automática ante fallos:

| **Servicio** | **Nombre del contenedor** | **Política de reinicio** | **Impacto** |
| --- | --- | --- | --- |
| **sqlserver** | **sqlserver** | **always** | Persistencia de la base de datos mantenida a través **sql\_data**del volumen |
| **sql-init** | (sin nombre) | Política de no reinicio | Se ejecuta una vez y luego sale. |
| **backend** | **backend** | **always** | El servicio API se recupera automáticamente |
| **frontend** | **frontend** | **always** | La interfaz web se recupera automáticamente |

El **sql-init**servicio está diseñado para ejecutarse una vez durante el inicio y no tiene una política de reinicio, ya que la inicialización de la base de datos solo debe ocurrir durante la implementación inicial.

**Redes y almacenamiento**

Esta página documenta la configuración de red y almacenamiento persistente del sistema de registro de personas en contenedores Deploy-Analisis-Registros. Abarca la topología de red de Docker Compose, la asignación de puertos para acceso externo y la gestión de volúmenes para la persistencia de datos.

**Arquitectura de red**

El sistema utiliza una única red Docker denominada "Docker" **backend**para permitir la comunicación segura entre contenedores. Los cuatro servicios están conectados a esta red, lo que les permite comunicarse utilizando nombres de contenedor como nombres de host.

**Comunicación de contenedores**

Los servicios se comunican internamente utilizando nombres de contenedores como nombres de host DNS dentro de la **backend**red:

| **Contenedor de origen** | **Contenedor de destino** | **Método de comunicación** |
| --- | --- | --- |
| **frontend** | **backend** | Solicitudes HTTP a**backend:5035** |
| **backend** | **sqlserver** | Conexiones SQL a**sqlserver:1433** |
| **sql-init** | **sqlserver** | Comandos SQL para**sqlserver:1433** |

La configuración de la red se define de forma mínima en el archivo de composición, basándose en el comportamiento de la red de puente predeterminado de Docker con descubrimiento de servicio.

**Asignaciones de puertos y acceso externo**

El sistema expone tres servicios a clientes externos a través de asignaciones de puertos de host:

**Detalles del puerto de servicio**

| **Servicio** | **Puerto externo** | **Puerto interno** | **Protocolo** | **Objetivo** |
| --- | --- | --- | --- | --- |
| **frontend** | 4200 | 80 | HTTP | Acceso a aplicaciones web |
| **backend** | 5035 | 5035 | HTTP | Puntos finales de la API REST |
| **sqlserver** | 1433 | 1433 | SQL | Conexiones de base de datos |

El **sql-init**servicio no tiene puertos expuestos ya que realiza una inicialización única y sale.

**Configuración de almacenamiento**

El sistema implementa dos tipos de almacenamiento persistente: volúmenes Docker para datos de base de datos y montajes de enlace para scripts de inicialización.

**Especificaciones de volumen**

**Volumen nombrado:sql\_data**

* **Propósito** : Almacenamiento persistente para archivos de base de datos de SQL Server
* **Punto de montaje** : **/var/opt/mssql**en **sqlserver**contenedor
* **Contenido** : Archivos de base de datos, registros de transacciones, bases de datos del sistema
* **Ciclo de vida** : persiste a pesar de los reinicios y reconstrucciones del contenedor

**Montura de enlace:./script**

* **Propósito** : Proporciona acceso a scripts de inicialización
* **Fuente** : **./script**directorio relativo a**docker-compose.yml**
* **Punto de montaje** : **/scripts**en **sql-init**contenedor
* **Contenido** : **init.sql**script de inicialización de la base de datos
* **Acceso** : Acceso de solo lectura para la ejecución del script

**Patrones de uso del almacenamiento**

| **Recipiente** | **Tipo de volumen** | **Sendero del Monte** | **Patrón de uso** |
| --- | --- | --- | --- |
| **sqlserver** | Volumen nombrado | **/var/opt/mssql** | Operaciones de lectura y escritura de bases de datos |
| **sql-init** | Montaje de enlace | **/scripts** | Ejecución de scripts de solo lectura |

La configuración de almacenamiento garantiza la persistencia de la base de datos al tiempo que proporciona acceso a los scripts de inicialización durante la fase de configuración.

**Diseño de bases de datos**

**Propósito y alcance**

Este documento ofrece una descripción general completa del **RegistroPersona**diseño de la base de datos que sirve como almacén central de datos para el sistema de análisis del registro de personas. La base de datos está diseñada para gestionar registros individuales con clasificaciones geográficas y categorizaciones por tipo de persona, lo que respalda la funcionalidad principal del sistema: la gestión y el análisis del registro de personas. .

**Descripción general de la arquitectura de la base de datos**

La **RegistroPersona**base de datos implementa un diseño relacional normalizado con tres entidades centrales que respaldan los requisitos del sistema de registro de personas para almacenar datos individuales completos con clasificaciones geográficas y categóricas.

**Estructura de las tablas principales**

La base de datos consta de tres tablas principales que implementan un sistema de clasificación jerárquica de personas:

| **Mesa** | **Objetivo** | **Características clave** |
| --- | --- | --- |
| **Region** | Clasificación geográfica | Tabla de búsqueda simple con ID de incremento automático |
| **TipoPersona** | Categorización de tipos de personas | Restricción única en el campo de nombre |
| **Persona** | Registros de personas individuales | Entidad compleja con múltiples tipos de datos y restricciones |

**Patrón de relación de tablas**

El diseño sigue un patrón de esquema en estrella donde <i> **Persona**funciona</i> como tabla de hechos central, con <i> **Region**y **TipoPersona**</i> como tablas de dimensiones. Esta estructura facilita la consulta eficiente de datos personales filtrados por ubicación geográfica o tipo de persona.

**Decisiones clave de diseño:**

* **Claves primarias UUID** : la **Persona**tabla utiliza **VARCHAR(36)**de forma **NEWID()**predeterminada identificadores únicos globales
* **Claves externas que aceptan valores nulos** : las asignaciones regionales y de tipo son opcionales, lo que permite registros incompletos
* **Actualizaciones en cascada** : Las restricciones de clave externa se incluyen **ON UPDATE CASCADE**para flexibilidad de mantenimiento
* **SET NULL al eliminar** : conserva los registros de personas cuando se eliminan los valores de búsqueda

**Marco de integridad de datos**

La base de datos implementa múltiples capas de validación de datos para garantizar la consistencia y la calidad:

**Tipos de restricciones**

| **Categoría de restricción** | **Implementación** | **Objetivo** |
| --- | --- | --- |
| **Restricciones de clave principal** | ID de incremento automático, generación de UUID | Identificación única de registros |
| **Restricciones de clave externa** | **FK\_Persona\_Region**,**FK\_Persona\_TipoPersona** | Integridad referencial |
| **Comprobar restricciones** | **CHK\_TipoSangre**,**CHK\_Estado** | Validación de dominio |
| **Restricciones únicas** | **TipoPersona.Nombre UNIQUE** | Evitar categorías duplicadas |

**Reglas de validación**

El sistema aplica una validación estricta en los campos críticos:

* **Validación del tipo de sangre** : Restringido a los valores estándar del sistema ABO/Rh ( **A+**, **A-**, **B+**, **B-**, **AB+**, **AB-**, **O+**, **O-**)
* **Validación de estado** : Los registros de personas deben tener el estado **Confirmado**, **Pendiente**o**Vencido**
* **Campos obligatorios** : Nombre, apellido, fecha de nacimiento y fecha de residencia son obligatorios.

**Inicialización y propagación de bases de datos**

El proceso de creación de la base de datos sigue un enfoque transaccional con manejo integral de errores y población inicial de datos.

**Población de datos inicial**

El script de inicialización siembra la base de datos con datos de referencia fundamentales:

* **Regiones** : Cinco zonas geográficas ( **Norte**, **Sur**, **Este**, **Oeste**, **Centro**)
* **Tipos de persona** : Tres categorías de clasificación ( **Agente**, **Civil**, **Administrador**)
* **Usuarios de muestra** : Dos registros de prueba para la validación del sistema

Esta estrategia de siembra garantiza que el sistema tenga datos de referencia operativos inmediatamente después de la implementación mientras mantiene la idempotencia a través de verificaciones de existencia.

**Optimización del rendimiento**

La base de datos incluye indexación específica para patrones de consulta comunes:

* **Índice compuesto** : **IX\_Persona\_Nombre**admite **[Primer Nombre], [Primer Apellido]**búsquedas basadas en nombres
* **Índices de clave principal** : indexación automática en todos los campos de clave principal
* **Rendimiento de clave externa** : las claves externas enteras proporcionan uniones eficientes a las tablas de búsqueda

La estrategia de indexación equilibra el rendimiento de las consultas con la eficiencia del almacenamiento, centrándose en los patrones de acceso más comunes para búsquedas de nombres de personas y uniones de relaciones.

**Esquema y tablas**

Este documento proporciona documentación detallada del esquema principal de la base de datos RegistroPersona, incluyendo estructuras de tablas, definiciones de columnas, tipos de datos y configuraciones de claves primarias. El esquema consta de tres tablas principales que constituyen la base del sistema de registro de personas: **Region**, **TipoPersona**, y **Persona**..

**Descripción general de la estructura de la base de datos**

La **RegistroPersona**base de datos implementa un esquema normalizado diseñado para almacenar datos completos de registros de personas con organización geográfica y categórica. El esquema sigue los principios estándar de diseño de bases de datos relacionales, con tablas de búsqueda para regiones y tipos de personas, y una **Persona**tabla central que contiene registros individuales.

**Estructura del esquema de base de datos**

**Tabla de regiones**

La **Region**tabla sirve como tabla de consulta para las regiones geográficas dentro del sistema de registro de personas. Proporciona clasificaciones regionales estandarizadas para organizar los registros de personas por ubicación.

| **Columna** | **Tipo de datos** | **Restricciones** | **Descripción** |
| --- | --- | --- | --- |
| **Id** | **INT** | **IDENTITY(1,1) PRIMARY KEY** | Clave primaria de incremento automático |
| **Nombre** | **NVARCHAR(100)** | **NOT NULL** | Nombre de la región |

La tabla está rellena previamente con cinco regiones estándar: Norte, Sur, Este, Oeste y Centro, lo que proporciona una cobertura geográfica integral para el sistema de registro.

**Tabla TipoPersona**

La **TipoPersona**tabla define el sistema de clasificación categórica para los diferentes tipos de personas dentro del registro. Garantiza la coherencia en la asignación de tipos de personas en todo el sistema.

| **Columna** | **Tipo de datos** | **Restricciones** | **Descripción** |
| --- | --- | --- | --- |
| **Id** | **INT** | **IDENTITY(1,1) PRIMARY KEY** | Clave primaria de incremento automático |
| **Nombre** | **NVARCHAR(100)** | **NOT NULL UNIQUE** | Nombre del tipo de persona, debe ser único |

El sistema se inicializa con tres tipos de personas predefinidos:

* **Agente** : Clasificación del agente
* **Civil** : Clasificación civil
* **Administrador** : Clasificación de administrador

**Tabla de personas**

La **Persona**tabla es la entidad central que contiene los registros completos de cada persona. Almacena información personal, fechas, datos biológicos, asignaciones regionales y metadatos del sistema.

| **Columna** | **Tipo de datos** | **Restricciones** | **Descripción** |
| --- | --- | --- | --- |
| **Id** | **VARCHAR(36)** | **PRIMARY KEY DEFAULT NEWID()** | Identificador único basado en GUID |
| **[Primer Nombre]** | **NVARCHAR(50)** | **NOT NULL** | Nombre, campo obligatorio |
| **[Segundo Nombre]** | **NVARCHAR(50)** | **NULL** | Segundo nombre, opcional |
| **[Primer Apellido]** | **NVARCHAR(50)** | **NOT NULL** | Primer apellido, campo obligatorio |
| **[Segundo Apellido]** | **NVARCHAR(50)** | **NULL** | Segundo apellido, opcional |
| **[Fecha de nacimiento]** | **DATE** | **NOT NULL** | Fecha de nacimiento, requerida |
| **[Fecha de residencia]** | **DATE** | **NOT NULL** | Fecha de residencia, requerida |
| **[Tipo de sangre]** | **NVARCHAR(3)** | **CHK\_TipoSangre** | Tipo de sangre con restricción de comprobación |
| **RegionId** | **INT** | **NULL, FK\_Persona\_Region** | Clave externa a la tabla Región |
| **TipoPersonaId** | **INT** | **NULL, FK\_Persona\_TipoPersona** | Clave externa a la tabla TipoPersona |
| **Genero** | **NVARCHAR(20)** | **NULL** | Especificación de género |
| **Foto** | **VARBINARY(MAX)** | **NULL** | Almacenamiento de datos de fotografías binarias |
| **Estado** | **NVARCHAR(20)** | **CHK\_Estado** | Estado con restricción de verificación |

**Comprobar restricciones**

La **Persona**tabla implementa dos restricciones de verificación para la validación de datos:

* **CHK\_TipoSangre**: Valida los valores del tipo de sangre contra los tipos de sistema ABO estándar: **A+**, **A-**, **B+**, **B-**, **AB+**, **AB-**, **O+**,**O-**
* **CHK\_Estado**: Valida los valores de estado a: **Confirmado**, **Pendiente**,**Vencido**

**Estrategia de indexación**

La tabla incluye un índice compuesto **IX\_Persona\_Nombre**y columnas para optimizar las consultas basadas **[Primer Nombre]**en **[Primer Apellido]**nombres.

**Tipos de datos de columna y consideraciones de almacenamiento**

| **Mesa** | **Columna** | **Tipo de servidor SQL** | **Tamaño de almacenamiento** | **Objetivo** |
| --- | --- | --- | --- | --- |
| Región | Identificación | **INT IDENTITY** | 4 bytes | Clave principal, incremento automático |
| Región | Nombre | **NVARCHAR(100)** | 200 bytes máximo | Nombres de regiones Unicode |
| TipoPersona | Identificación | **INT IDENTITY** | 4 bytes | Clave principal, incremento automático |
| TipoPersona | Nombre | **NVARCHAR(100)** | 200 bytes máximo | Nombres de tipos de personas Unicode |
| Persona | Identificación | **VARCHAR(36)** | 36 bytes | Representación de cadena GUID |
| Persona | [Primer Nombre] | **NVARCHAR(50)** | 100 bytes máximo | Nombre Unicode |
| Persona | [Segundo nombre] | **NVARCHAR(50)** | 100 bytes máximo | Segundo nombre Unicode |
| Persona | [Primer Apellido] | **NVARCHAR(50)** | 100 bytes máximo | Primer apellido Unicode |
| Persona | [Segundo Apellido] | **NVARCHAR(50)** | 100 bytes máximo | Segundo apellido Unicode |
| Persona | [Fecha de nacimiento] | **DATE** | 3 bytes | Solo fecha de nacimiento |
| Persona | [Fecha de residencia] | **DATE** | 3 bytes | Solo fecha de residencia |
| Persona | [Tipo de sangre] | **NVARCHAR(3)** | 6 bytes máximo | Códigos de tipo de sangre |
| Persona | Identificación de región | **INT** | 4 bytes | Referencia de clave externa |
| Persona | TipoPersonaId | **INT** | 4 bytes | Referencia de clave externa |
| Persona | Género | **NVARCHAR(20)** | 40 bytes máximo | Designación de género |
| Persona | Foto | **VARBINARY(MAX)** | 2 GB máximo | Datos de imágenes binarias |
| Persona | Estado | **NVARCHAR(20)** | 40 bytes máximo | Designación de estado |

**Relaciones y restricciones**

**Relaciones de claves externas**

La base de datos RegistroPersona implementa un modelo relacional con dos relaciones de clave externa primaria que conectan la **Persona**tabla con las tablas de datos de referencia. Ambas relaciones utilizan **ON DELETE SET NULL**políticas **ON UPDATE CASCADE**para mantener la integridad referencial y preservar los registros de personas cuando se modifican los datos de referencia.

**Restricción de región de FK\_Persona**

La **FK\_Persona\_Region**restricción vincula a cada persona con su región geográfica mediante la **RegionId**columna. Esta relación permite al sistema rastrear la distribución regional de las personas registradas.

| **Propiedad de restricción** | **Valor** |
| --- | --- |
| Nombre de la restricción | **FK\_Persona\_Region** |
| Columna de clave externa | **Persona.RegionId** |
| Tabla/columna referenciada | **Region.Id** |
| Eliminar acción | **SET NULL** |
| Acción de actualización | **CASCADE** |

**Restricción FK\_Persona\_TipoPersona**

La **FK\_Persona\_TipoPersona**restricción categoriza a cada persona por su tipo (Agente, Civil o Administrador) a través de la **TipoPersonaId**columna. Esta clasificación permite la organización de datos basada en roles y el control de acceso.

| **Propiedad de restricción** | **Valor** |
| --- | --- |
| Nombre de la restricción | **FK\_Persona\_TipoPersona** |
| Columna de clave externa | **Persona.TipoPersonaId** |
| Tabla/columna referenciada | **TipoPersona.Id** |
| Eliminar acción | **SET NULL** |
| Acción de actualización | **CASCADE** |

**Comprobar restricciones**

La base de datos implementa dos restricciones de verificación en la **Persona**tabla para garantizar la calidad y la consistencia de los datos para los valores enumerados.

**Restricción CHK\_TipoSangre**

La **CHK\_TipoSangre**restricción valida los valores del tipo de sangre para garantizar que se ajusten a la notación del grupo sanguíneo ABO estándar con indicadores del factor Rh.

**Valores permitidos:**

* **A+**, **A-**- Sangre tipo A con factor Rh positivo/negativo
* **B+**, **B-**- Sangre tipo B con factor Rh positivo/negativo
* **AB+**, **AB-**- Sangre tipo AB con factor Rh positivo/negativo
* **O+**, **O-**- Sangre tipo O con factor Rh positivo/negativo

**Restricción CHK\_Estado**

La **CHK\_Estado**restricción impone valores de estado válidos para el seguimiento del estado de verificación de los registros de personas en el sistema de registro.

**Valores permitidos:**

* **Confirmado**- Registro de persona verificada y activa
* **Pendiente**- En espera de verificación o procesamiento
* **Vencido**- Registro vencido o inactivo

**Estrategia de índice**

La base de datos implementa una estrategia de indexación específica enfocada en optimizar las consultas basadas en nombres, que se espera que sean las operaciones de búsqueda más comunes para la funcionalidad de búsqueda de personas.

**Propiedades del índice:**

* **Nombre:** **IX\_Persona\_Nombre**
* **Tipo:** Índice compuesto no agrupado
* **Columnas:** **[Primer Nombre], [Primer Apellido]**
* **Propósito:** Optimizar las búsquedas basadas en nombres y las operaciones de clasificación

Este índice permite la ejecución eficiente de consultas que filtran u ordenan por nombre o por la combinación de nombre y apellido, que son los patrones más probables de búsqueda de personas en el sistema de registro.

**Reglas de integridad de datos**

La base de datos implementa un conjunto integral de reglas de integridad de datos a través de claves primarias, restricciones únicas y políticas de manejo de valores nulos que garantizan la consistencia de los datos en todo el sistema de registro de personas.

**Restricciones únicas**

| **Mesa** | **Columna** | **Propósito de la restricción** |
| --- | --- | --- |
| **TipoPersona** | **Nombre** | Evita nombres de tipos de personas duplicados |

**Política de manejo de valores nulos**

La base de datos implementa un manejo selectivo de valores nulos en función de la criticidad de los datos:

**Campos obligatorios (NO NULOS):**

* Identificación de la persona: **Primer Nombre**,**Primer Apellido**
* Datos temporales: **Fecha de nacimiento**,**Fecha de residencia**
* Datos de referencia: **Region.Nombre**,**TipoPersona.Nombre**

**Campos opcionales (se permite NULL):**

* Nombres secundarios: **Segundo Nombre**,**Segundo Apellido**
* Datos de clasificación: **RegionId**, **TipoPersonaId**, **Genero**,**Estado**
* Datos binarios:**Foto**
* Datos médicos:**Tipo de sangre**

**Reglas de cascada de integridad referencial**

Ambas restricciones de clave externa utilizan políticas en cascada idénticas:

* **AL ELIMINAR ESTABLECER EN NULL:** Cuando se elimina un registro referenciado, el valor de la clave externa se establece en NULL en lugar de evitar la eliminación o ejecutar la eliminación en cascada.
* **EN CASCADA DE ACTUALIZACIONES:** Cuando se actualiza una clave principal referenciada, todos los valores de clave externa se actualizan automáticamente para mantener la coherencia

Este enfoque preserva los registros de personas incluso cuando se modifican los datos de referencia, evitando la pérdida de datos y manteniendo la integridad referencial.

**Inicialización de la base de datos**

Esta página documenta el proceso de inicialización de la base de datos que ocurre durante el arranque del sistema, específicamente la creación de la **RegistroPersona**base de datos, sus tablas, restricciones, índices y la inicialización de datos. La inicialización se gestiona mediante el **init.sql**script ejecutado por el servicio sql-init.

**Descripción general de la inicialización**

El proceso de inicialización de la base de datos se implementa como un único script SQL que se ejecuta dentro de una transacción para garantizar la atomicidad. El script está diseñado para ser idempotente, lo que le permite ejecutarse varias veces sin causar errores ni duplicación de datos.

**Gestión de bases de datos y contexto**

El proceso de inicialización comienza estableciendo el contexto de base de datos adecuado y garantizando que la **RegistroPersona**base de datos exista.

**Lógica de creación de bases de datos**

| **Paso** | **Operación** | **Implementación** |
| --- | --- | --- |
| 1 | Cambiar al contexto maestro | **USE master; GO** |
| 2 | Comprobar la existencia de la base de datos | **SELECT name FROM sys.databases WHERE name = 'RegistroPersona'** |
| 3 | Crear si no existe | **CREATE DATABASE RegistroPersona** |
| 4 | Cambiar a nueva base de datos | **USE RegistroPersona; GO** |

El script utiliza lógica condicional para evitar intentar crear una base de datos que ya existe, imprimiendo mensajes de estado apropiados para cada escenario.

**Implementación de la creación de tablas**

Cada creación de tabla sigue el patrón:

1. Comprobar la existencia de la tabla utilizando**sys.tables**
2. Crear tabla con esquema completo si no existe
3. Mensaje de confirmación de impresión

La **Persona**tabla incluye restricciones integrales y relaciones de clave externa establecidas durante la creación en lugar de agregarse posteriormente.

**Estrategia de siembra de datos**

El script de inicialización genera datos de referencia necesarios para el funcionamiento del sistema y crea usuarios de muestra para fines de prueba.

**Siembra de datos de referencia**

| **Mesa** | **Datos** | **Implementación** |
| --- | --- | --- |
| **TipoPersona** | Agente, Civil, Administrador | INSERTAR condicional con comprobación de existencia |
| **Region** | Norte, Sur, Este, Oeste, Centro | INSERTAR por lotes si la tabla está vacía |

**Gestión de transacciones y manejo de errores**

El script de inicialización implementa una gestión integral de transacciones con manejo de errores para garantizar la integridad de los datos.

**Implementación del manejo de errores**

El script utiliza el mecanismo TRY/CATCH de SQL Server para manejar errores:

| **Componente** | **Objetivo** | **Implementación** |
| --- | --- | --- |
| **BEGIN TRY** | Iniciar bloque protegido | Envuelve todas las operaciones DDL y DML |
| **BEGIN TRANSACTION** | Iniciar transacción | Garantiza la atomicidad de todas las operaciones. |
| **COMMIT TRANSACTION** | Finalizar cambios | Se ejecuta solo si todas las operaciones tienen éxito |
| **BEGIN CATCH** | Manejador de errores | Detecta cualquier excepción durante la ejecución |
| **ROLLBACK TRANSACTION** | Deshacer cambios | Revierte todos los cambios si ocurre un error |
| Informe de errores | Información de diagnóstico | Imprime ERROR\_MESSAGE(), ERROR\_NUMBER(), ERROR\_LINE() |

La verificación de transacciones **IF @@TRANCOUNT > 0**garantiza que la reversión solo ocurra si una transacción está activa, lo que evita errores adicionales en el controlador de errores.

[script/init.sql152-160](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/script/init.sql#L152-L160)

**Patrones de diseño idempotentes**

El script implementa varios patrones para garantizar una reejecución segura:

| **Patrón** | **Implementación** | **Objetivo** |
| --- | --- | --- |
| Comprobación de existencia de la base de datos | **IF NOT EXISTS (SELECT name FROM sys.databases...)** | Prevenir errores de creación de bases de datos |
| Comprobación de existencia de tabla | **IF NOT EXISTS (SELECT \* FROM sys.tables...)** | Prevenir errores de creación de tablas |
| Comprobación de existencia de datos | **IF NOT EXISTS (SELECT 1 FROM TableName WHERE...)** | Evitar la inserción de datos duplicados |
| Comprobación de existencia de índice | **IF NOT EXISTS (SELECT \* FROM sys.indexes...)** | Prevenir errores de creación de índices |

Este diseño permite que el servicio sql-init ejecute el script varias veces sin causar fallas, lo que admite escenarios de reinicio de contenedores y flujos de trabajo de desarrollo.

**Guía de implementación**

Este documento proporciona instrucciones completas para implementar el sistema de registro de personas Deploy-Analisis-Registros mediante Docker Compose. Abarca todo el proceso de implementación, desde los prerrequisitos hasta la verificación del servicio, incluyendo las opciones de configuración y los pasos para la solución de problemas.

**Prerrequisitos y configuración del entorno**

Antes de implementar el sistema de registro de personas, asegúrese de que se cumplan los siguientes requisitos:

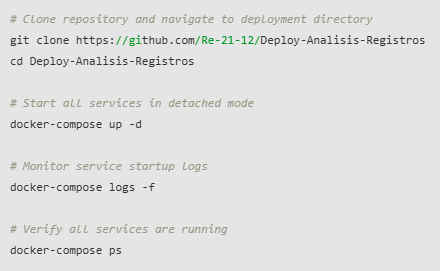
| **Requisito** | **Versión** | **Objetivo** |
| --- | --- | --- |
| Estibador | 20.10+ | Tiempo de ejecución del contenedor |
| Docker Compose | 3.8+ | Orquestación de servicios |
| Puertos disponibles | 1433, 4200, 5035 | SQL Server, interfaz, back-end |
| Espacio en disco | 2 GB+ | Imágenes de contenedores y almacenamiento de bases de datos |
| Memoria | 4 GB+ | SQL Server y servicios de aplicación |

El sistema requiere conectividad a Internet durante la implementación inicial para extraer las siguientes imágenes de contenedor:

* **mcr.microsoft.com/mssql/server:2022-latest**
* **mcr.microsoft.com/mssql-tools**
* **revic2112/dev-analisis-backend:latest**
* **revic2112/dev-analisis-front:latest**

**Comandos de inicio rápido**

Implemente el sistema completo con estos comandos:

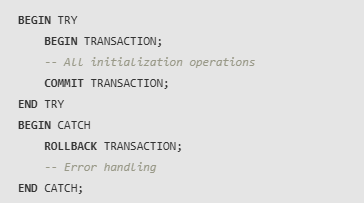


**Proceso de inicialización de la base de datos**

El **sql-init**servicio ejecuta la configuración de la base de datos automáticamente durante la implementación:

| **Paso de inicialización** | **Acción** | **Ubicación en init.sql** |
| --- | --- | --- |
| Creación de bases de datos | **CREATE DATABASE RegistroPersona** | [script/init.sql6-10](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/script/init.sql#L6-L10) |
| Creación de esquemas | Crear **Region**, **TipoPersona**, **Persona**tablas | [script/init.sql25-97](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/script/init.sql#L25-L97) |
| Configuración de restricciones | Agregar claves externas y verificar restricciones | [script/init.sql79-94](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/script/init.sql#L79-L94) |
| Creación de índices | **IX\_Persona\_Nombre**en los campos de nombre | [script/init.sql100-104](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/script/init.sql#L100-L104) |
| Datos de semillas | Insertar regiones y tipos de personas | [script/init.sql107-148](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/script/init.sql#L107-L148) |
| Usuarios de prueba | Crear muestra **Agente**y **Administrador**usuarios | [script/init.sql115-148](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/script/init.sql#L115-L148) |

El script de inicialización utiliza la gestión de transacciones para garantizar la consistencia de los datos:



**Variables de entorno y configuración**

El servicio SQL Server requiere variables de entorno específicas para una inicialización adecuada:

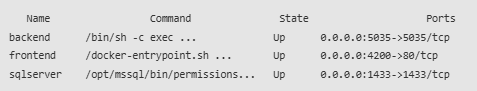
| **Variable** | **Valor** | **Objetivo** | **Ubicación de la configuración** |
| --- | --- | --- | --- |
| **ACCEPT\_EULA** | **Y** | Aceptar la licencia de SQL Server | [docker-compose.yml31](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L31-L31) |
| **SA\_PASSWORD** | **YourStrong!Passw0rd** | Contraseña del administrador del sistema | [docker-compose.yml32](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L32-L32) |
| **MSSQL\_PID** | **Express** | Edición SQL Server | [docker-compose.yml33](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L33-L33) |

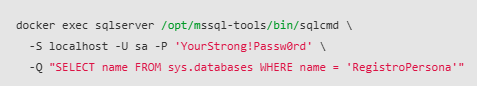
**Verificación del estado del servicio**

Después de la implementación, verifique que cada servicio funcione correctamente:

**1. Comprobación del estado del contenedor**



El resultado esperado muestra todos los servicios en estado "Activo":

**2. Prueba de conectividad de la base de datos**

**3. Accesibilidad de la aplicación**

* Base de datos: Conéctese mediante SQL Server Management Studio o Azure Data Studio

**Solución de problemas comunes**

**Errores de inicio de SQL Server**

* **Problema**: El contenedor sale con errores de autenticación
* **Solución**: Verifique **SA\_PASSWORD** que cumpla con los requisitos de complejidad (más de 8 caracteres, mayúsculas, minúsculas, dígitos, símbolos)
* **Controlar**: docker**-compose logs sqlserver**

**Errores de inicialización de la base de datos**

* **Problema** : **sql-init**el contenedor no puede ejecutar el script
* **Solución** : asegúrese de que **./script/init.sql**el archivo exista y tenga finales de línea adecuados (LF, no CRLF)
* **Controlar** :**docker-compose logs sql-init**

**Conflictos portuarios**

* **Problema** : Errores de "Puerto ya en uso" durante el inicio
* **Solución** : Modificar las asignaciones de puertos en[docker-compose.yml9](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L9-L9) [docker-compose.yml20](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L20-L20) [docker-compose.yml35](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L35-L35)

**Problemas de montaje de volumen**

* **Problema**: Los datos de la base de datos no persisten entre reinicios
* **Solución**: Verifique que Docker tenga permisos para crear volúmenes en el directorio actual

**Consideraciones sobre la implementación de producción**

Para entornos de producción, considere estas modificaciones:

1. **Seguridad**: cambie las contraseñas predeterminadas y use secretos de Docker
2. **Persistencia**: configurar montajes de volúmenes externos para el almacenamiento de bases de datos
3. **Redes**: utilice redes personalizadas con acceso restringido
4. **Monitoreo**: agregue controles de estado y configuración de registro
5. **Escalabilidad**: considere usar Docker Swarm o Kubernetes para la orquestación

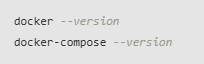
**Inicio rápido**

Esta página proporciona instrucciones paso a paso para implementar y ejecutar el sistema de registro de personas Deploy-Analisis-Registros localmente mediante Docker Compose. Abarca los comandos esenciales para que los cuatro servicios en contenedores estén operativos y accesibles.

**Prerrequisitos**

Antes de comenzar la implementación, asegúrese de que se cumplan los siguientes requisitos:

| **Requisito** | **Versión** | **Objetivo** |
| --- | --- | --- |
| Motor Docker | 20.10+ | Tiempo de ejecución del contenedor |
| Docker Compose | 2.0+ | Orquestación de servicios |
| Puertos disponibles | 1433, 4200, 5035 | Accesibilidad del servicio |
| Memoria disponible | 4 GB+ | Requisitos de SQL Server |

**Verificar la instalación de Docker:**

**Pasos de implementación**

**Paso 1: Clonar y navegar al repositorio**

**Paso 2: Iniciar todos los servicios**

Ejecute el comando de implementación de Docker Compose:



**Paso 3: Verificar el estado del servicio**

Compruebe que todos los contenedores estén en ejecución:

El resultado esperado debería mostrar cuatro contenedores en estado "Activo":

| **Nombre del contenedor** | **Servicio** | **Estado** | **Puertos** |
| --- | --- | --- | --- |
| **sqlserver** | SQL Server 2022 | Arriba | 0.0.0.0:1433->1433/tcp |
| **backend** | Servicio API | Arriba | 0.0.0.0:5035->5035/tcp |
| **frontend** | Aplicación web | Arriba | 0.0.0.0:4200->80/tcp |
| **sql-init** | Inicialización de la base de datos | Salió (0) | - |

**Nota** : El **sql-init**contenedor debe mostrar el estado "Salido (0)", lo que indica que la inicialización de la base de datos fue exitosa.

**Puntos de acceso al servicio**

Una vez completada la implementación, los componentes del sistema serán accesibles a través de los siguientes puntos finales:

**Detalles de la conexión a la base de datos**

Para acceder directamente a la base de datos mediante SQL Server Management Studio o herramientas similares:

| **Parámetro** | **Valor** |
| --- | --- |
| **Servidor** | **localhost,1433** |
| **Autenticación** | Autenticación de SQL Server |
| **Acceso** | **sa** |
| **Contraseña** | **YourStrong!Passw0rd** |
| **Base de datos** | **RegistroPersona** |

**Verificación de la base de datos**

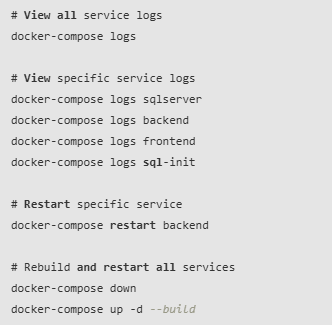
Verifique que la inicialización de la base de datos se haya completado exitosamente verificando las tablas creadas:

Busque estos mensajes de éxito en la salida:

* "Base de datos RegistroPersona creada exitosamente."
* "Tabla Región creada exitosamente."
* "Tabla TipoPersona creada exitosamente."
* "Tabla Persona creada exitosamente."

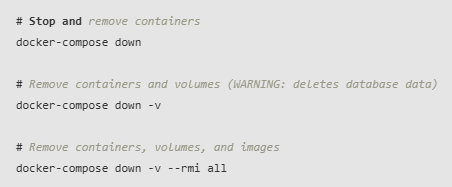
**Comandos de estado del servicio**

Consultar el estado del servicio individual:



**Comandos de limpieza**

Para eliminar completamente la implementación:



**Configuración y entorno**

Esta página documenta la configuración y los ajustes del entorno del sistema de registro de personas Deploy-Analisis-Registros. Abarca las variables de entorno, las configuraciones de servicio, la configuración de red y las opciones de personalización para diferentes escenarios de implementación.

**Configuración de variables de entorno**

El sistema utiliza varias variables de entorno para configurar el servicio de base de datos de SQL Server y controlar el comportamiento de la aplicación.

**Variables de entorno de SQL Server**

| **Variable** | **Valor** | **Objetivo** |
| --- | --- | --- |
| **ACCEPT\_EULA** | **Y** | Acepta el Acuerdo de licencia de usuario final de Microsoft SQL Server |
| **SA\_PASSWORD** | **YourStrong!Passw0rd** | Establece la contraseña de administrador del sistema para SQL Server |
| **MSSQL\_PID** | **Express** | Especifica la edición de SQL Server (Express para desarrollo) |

**Configuración del puerto de servicio**

El sistema expone tres puertos principales para acceso externo, configurados a través de asignaciones de puertos de Docker Compose.

**Configuración de mapeo de puertos**

| **Servicio** | **Puerto de contenedores** | **Puerto host** | **Objetivo** |
| --- | --- | --- | --- |
| **frontend** | **80** | **4200** | Acceso a aplicaciones web |
| **backend** | **5035** | **5035** | Puntos finales de la API REST |
| **sqlserver** | **1433** | **1433** | Acceso directo a la base de datos |

**Configuración de volumen y almacenamiento**

El sistema utiliza volúmenes Docker para la persistencia de datos y el montaje de scripts.

**Definiciones de volumen**

| **Tipo de volumen** | **Nombre del volumen** | **Punto de montaje** | **Objetivo** |
| --- | --- | --- | --- |
| Volumen nombrado | **sql\_data** | **/var/opt/mssql** | Persistencia de datos de SQL Server |
| Montura de enlace | **./script** | **/scripts** | Scripts de inicialización de bases de datos |

**Configuración de red**

Todos los servicios se comunican a través de una red Docker personalizada denominada **backend**.

**Comunicación de servicios de red**

La **backend**red definida en[docker-compose.yml55-56](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L55-L56)permite la comunicación interna de servicio a servicio utilizando nombres de contenedores como nombres de host.

| **Servicio** | **Nombre del contenedor** | **Rol de red** |
| --- | --- | --- |
| **sqlserver** | **sqlserver** | Servicio de base de datos |
| **sql-init** | Generado automáticamente | Inicialización de la base de datos |
| **backend** | **backend** | Servicio API |
| **frontend** | **frontend** | Aplicación web |

**Configuración de la conexión a la base de datos**

**Componentes de la cadena de conexión**

La conexión a SQL Server se establece utilizando estos parámetros en el **sql-init**punto de entrada del servicio[docker-compose.yml47](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L47-L47):

* **Servidor:** **sqlserver** (resolución de nombre de contenedor)
* **Nombre de usuario:** **sa** (administrador del sistema)
* **Contraseña:** **YourStrong!Passw0rd** (de la variable de entorno)
* **Script de entrada:** **/scripts/init.sql**

**Configuración del tiempo de inicialización**

El **sql-init**servicio incluye un retraso de 15 segundos ( **sleep 15**) para garantizar que SQL Server esté completamente listo antes de intentar la conexión.

**Configuración de seguridad**

**Seguridad de la contraseña**

El servidor SQL predeterminado **SA\_PASSWORD**está configurado **YourStrong!Passw0rd**en[docker-compose.yml32](https://github.com/Re-21-12/Deploy-Analisis-Registros/blob/7351401e/docker-compose.yml#L32-L32)Esta contraseña cumple con los requisitos de complejidad de Microsoft:

* Mínimo 8 caracteres
* Contiene letras mayúsculas
* Contiene letras minúsculas
* Contiene números
* Contiene caracteres especiales

**Consideraciones de seguridad para la producción**

Para implementaciones de producción, considere:

1. **Externalización de variables de entorno:** utilizar**. env**archivos o secretos de Docker
2. **Rotación de contraseñas:** Implemente cambios de contraseña regulares
3. **Aislamiento de red:** restringir la exposición del puerto de la base de datos
4. **Control de acceso de usuarios:** crear usuarios de bases de datos específicos de la aplicación

**Personalización para diferentes entornos**

**Modificaciones específicas del entorno**

**Entorno de desarrollo**

* Mantener la configuración predeterminada tal como está
* Puerto de base de datos **1433**expuesto para acceso directo
* Edición Express adecuada para cargas de trabajo de desarrollo

**Entorno de producción**

Modifique estos elementos de configuración:

| **Componente** | **Desarrollo** | **Producción** |
| --- | --- | --- |
| **SA\_PASSWORD** | **YourStrong!Passw0rd** | Gestión de secretos externos |
| **MSSQL\_PID** | **Express** | **Standard**o**Enterprise** |
| Puerto**1433** | Expuesto | Solo para uso interno |
| **restart** | **always** | **unless-stopped** |

**Entorno de prueba**

* Utilice nombres de volúmenes separados para evitar conflictos de datos
* Considere usar volúmenes efímeros para ejecuciones de pruebas limpias
* Modificar las asignaciones de puertos para evitar conflictos con instancias de desarrollo

**Recursos de desarrollo**

Esta página documenta los artefactos de desarrollo y los materiales de apoyo creados durante la evolución del sistema de registro de personas Deploy-Analisis-Registros. Abarca borradores de esquemas, iteraciones de diseño y documentación que ofrecen información sobre el proceso de desarrollo y el historial de toma de decisiones.

**Descripción general de la evolución del esquema**

El esquema de la base de datos experimentó una evolución significativa desde el borrador inicial hasta la implementación final, con agregados sustanciales en modelado de entidades, restricciones de integridad de datos y aprovisionamiento de datos semilla.

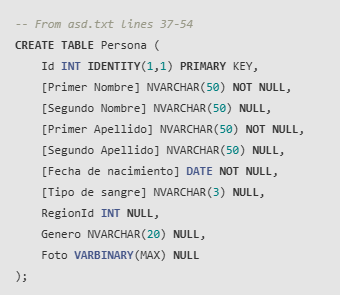
**Resumen de cambios clave en el esquema**

| **Aspecto** | **Versión borrador (asd.txt)** | **Versión final (init.sql)** |
| --- | --- | --- |
| **Estrategia de clave principal** | **INT IDENTITY(1,1)**para Persona | **VARCHAR(36) DEFAULT NEWID()**para Persona |
| **Recuento de entidades** | 2 mesas (Región, Persona) | 3 tablas (Región, TipoPersona, Persona) |
| **Clasificación de personas** | Ninguno | TipoPersona con tipos Agente, Civil, Administrador |
| **Campos adicionales** | Datos personales básicos | Ampliado con fecha de residencia, estado, tipo de persona |
| **Comprobar restricciones** | Solo tipo de sangre (con NULL) | Validación de tipo sanguíneo + estado |
| **Datos de semillas** | Solo datos de la región | Datos de semillas completos con usuarios de muestra |
| **Manejo de errores** | Incompleto (cortado) | Transacción completa con reversión |

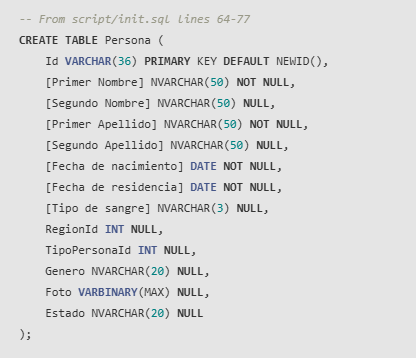
**Evolución del diseño de entidades**

**Transformación de la tabla Persona**

La **Persona**tabla sufrió los cambios más significativos entre el borrador y la versión final:

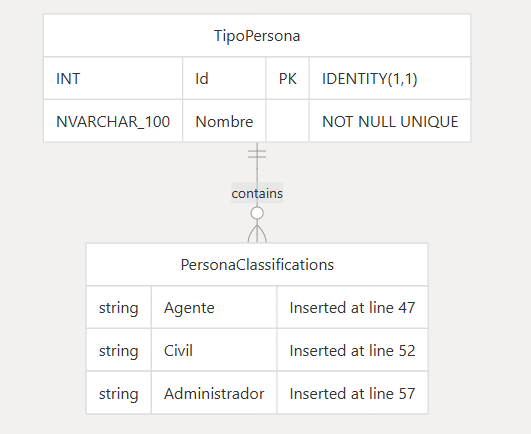
**Borrador de la estructura del personaje:**

**Estructura final del personaje:**



**Nueva entidad incorporada: TipoPersona**

El esquema final introdujo la **TipoPersona**entidad para soportar la clasificación de personas basada en roles:



**Perspectivas de desarrollo**

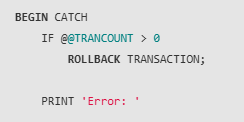
**Cambio de estrategia de clave principal**

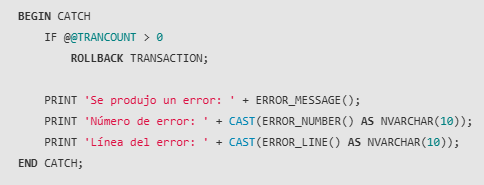
La migración de **INT IDENTITY**a **VARCHAR(36) DEFAULT NEWID()**indica un cambio hacia identificadores únicos globales, probablemente por compatibilidad de sistemas distribuidos o por requisitos de integración de API externas.

**Maduración del manejo de errores**

La versión borrador contiene un manejo de errores incompleto que corta la declaración a mitad de camino en la línea 85, mientras que la versión final implementa una reversión de transacciones integral con informes de errores detallados que incluyen el número de error y la información de la línea.

**Manejo de errores del borrador (incompleto):**



**Manejo final de errores (completo):**

**Evolución del patrón de diseño**

La evolución demuestra una progresión desde un modelo simple de dos entidades a un diseño más sofisticado de tres entidades con normalización adecuada, integridad de datos mejorada y datos semilla completos para una usabilidad inmediata del sistema.

**Evolución del esquema y borradores**

Este documento compara el borrador del esquema de la base de datos encontrado en [año faltante] **asd.txt**con la implementación final en [año faltante] **script/init.sql**, documentando la evolución del diseño de la base de datos del sistema RegistroPersona. Destaca las decisiones arquitectónicas, las características adicionales y las mejoras estructurales implementadas durante el proceso de desarrollo.

**Descripción general de la evolución del esquema**

El esquema de la base de datos experimentó una evolución significativa desde el borrador inicial hasta la implementación final. El borrador contenía un diseño simplificado de dos tablas, mientras que la versión final se expandió a una estructura más completa de tres tablas con restricciones, relaciones y capacidades de propagación de datos mejoradas.

**Cambios estructurales clave**

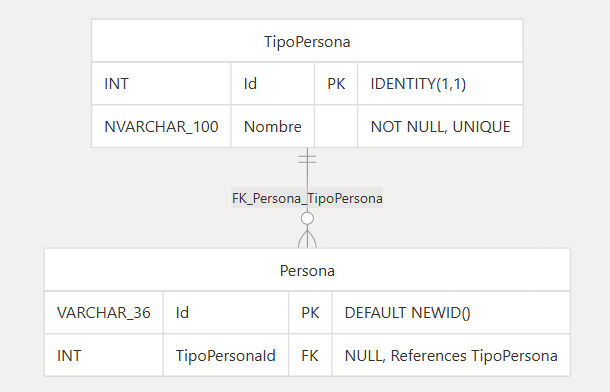
**Evolución de la estrategia de clave principal**

El cambio arquitectónico más significativo fue la modificación de la **Persona**estrategia de clave principal de la tabla:

| **Aspecto** | **Versión borrador** | **Versión final** |
| --- | --- | --- |
| **Tipo de clave principal** | **INT IDENTITY(1,1)** | **VARCHAR(36) DEFAULT NEWID()** |
| **Estrategia** | números enteros secuenciales | Identificadores basados ​​en GUID |
| **Escalabilidad** | Limitado a una sola base de datos | Admite sistemas distribuidos |
| **Unicidad** | Alcance de la base de datos | Único a nivel mundial |

**Introducción a la clasificación TipoPersona**

El esquema final introdujo una nueva **TipoPersona** tabla que estaba completamente ausente del borrador:



**Nuevos valores de clasificación:**

* **Agente**- Agentes del sistema
* **Civil**- Personal civil
* **Administrador**- Usuarios administrativos

**Adiciones y mejoras de columnas**

**Columnas faltantes en el borrador**

La implementación final agregó varias columnas críticas que no estaban presentes en el borrador:

| **Nombre de la columna** | **Tipo de datos** | **Objetivo** | **Restricción** |
| --- | --- | --- | --- |
| **[Fecha de residencia]** | **DATE NOT NULL** | Fecha de inicio de la residencia de seguimiento | Campo obligatorio |
| **TipoPersonaId** | **INT NULL** | Clasificación de personas | FK a TipoPersona |
| **Estado** | **NVARCHAR(20) NULL** | Seguimiento del estado de los registros | Restricción de comprobación |

**Evolución de la inicialización de datos**

**Comparación de estrategias de siembra**

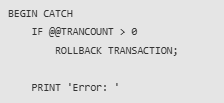
El enfoque de siembra de datos evolucionó significativamente entre versiones:

| **Componente** | **Borrador de implementación** | **Implementación final** |
| --- | --- | --- |
| **Datos de la región** | Sentencias INSERT básicas | INSERTAR condicional con comprobación de existencia |
| **Datos de TipoPersona** | No presente | Siembra completa con validación |
| **Usuarios de muestra** | Ninguno | Dos usuarios de prueba (Agente, Administrador) |
| **Gestión de transacciones** | Básico | Completo con manejo de errores |

**Manejo de errores y gestión de transacciones**

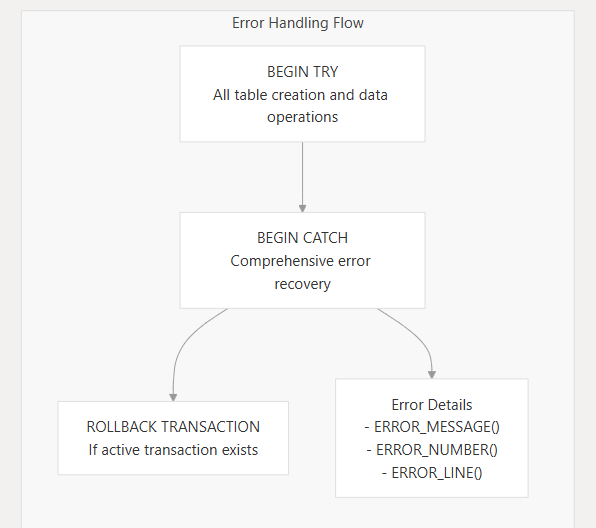
**Implementación del borrador incompleto**

La versión borrador contiene un manejo de errores incompleto que corta la declaración a mitad de camino:



**Manejo integral de errores finales**

La implementación final proporciona un manejo completo de errores con diagnósticos detallados:



**Implicaciones de la migración**

**Consideraciones sobre la actualización del esquema**

Las organizaciones que migren del esquema preliminar deberán abordar lo siguiente:

1. **Migración de clave principal** : convertir **INT IDENTITY**a **VARCHAR(36)**valores GUID
2. **Nueva adición de tabla** : crear y completar **TipoPersona**una tabla
3. **Adiciones de columnas** : agregue las columnas **[Fecha de residencia]**, **TipoPersonaId**, y**Estado**
4. **Actualizaciones de restricciones** : Implementar nuevas restricciones de verificación y relaciones de clave externa
5. **Clasificación de datos** : Asignar **TipoPersona**valores apropiados a los registros existentes

**Compatibilidad con versiones anteriores**

La evolución del esquema rompe la compatibilidad con versiones anteriores debido a:

* Cambio de tipo de clave principal que requiere actualizaciones del código de la aplicación
* Nuevas columnas obligatorias que necesitan valores predeterminados o migración de datos
* Restricciones de clave externa adicionales que requieren datos de referencia válidos