Sistema de Agendamiento de Citas ElectroHuila: Arquitectura de Base de Datos Oracle con Entity Framework Core y Arquitectura Limpia

Juan Esteban Huertas
SENA
Colombia, Colombia
jehuertas89@soy.sena.edu.co

Resumen

Este artículo presenta el diseño e implementación de la arquitectura de base de datos del sistema de agendamiento de citas para ElectroHuila, una empresa de servicios públicos que gestiona solicitudes de Peticiones, Quejas y Reclamos (PQR). El sistema emplea Oracle Database como gestor principal, implementado mediante Entity Framework Core 9.0 bajo los principios de Arquitectura Limpia y Domain-Driven Design. Se describe un esquema relacional de 22 tablas normalizadas a Tercera Forma Normal (3NF), optimizado para consultas de alto rendimiento mediante índices estratégicos y patrones de acceso a datos. La arquitectura contempla características avanzadas como soft deletes, auditoría temporal, conversión de tipos booleanos para Oracle, y convenciones de nomenclatura en mayúsculas. El diseño soporta múltiples gestores de bases de datos (Oracle, SQL Server, PostgreSQL, MySQL) mediante abstracciones del ORM, facilitando la portabilidad y escalabilidad del sistema. Los resultados demuestran una estructura robusta, mantenible y escalable que satisface los requerimientos de integridad referencial, seguridad de datos y rendimiento transaccional del dominio de agendamiento de citas.

Keywords

base de datos Oracle, Entity Framework Core, arquitectura limpia, diseño de esquemas, normalización, agendamiento de citas, sistemas PQR

1. Introducción

El sistema de agendamiento de citas ElectroHuila representa un caso de estudio significativo en el diseño de bases de datos para sistemas de gestión de Peticiones, Quejas y Reclamos (PQR) en empresas de servicios públicos. La complejidad inherente a este dominio requiere una arquitectura de datos robusta que soporte múltiples dimensiones: gestión de citas, usuarios con diferentes roles y permisos, sucursales distribuidas geográficamente, categorización de servicios, y trazabilidad completa de las solicitudes ciudadanas.

El diseño de la base de datos enfrentó varios desafíos técnicos fundamentales. Primero, la necesidad de modelar relaciones complejas entre 22 entidades del dominio, manteniendo la integridad referencial y la consistencia transaccional. Segundo, la implementación sobre Oracle Database, que requiere adaptaciones específicas como la conversión de tipos booleanos a NUMBER(1) y el uso de convenciones de nomenclatura en mayúsculas. Tercero, la aplicación de principios de Arquitectura Limpia y Domain-Driven Design, que demandan una separación clara entre el modelo de dominio y la persistencia física.

El esquema relacional diseñado contempla las siguientes áreas funcionales principales:

Gestión de Identidad y Acceso: Tablas para usuarios, roles, permisos, y asignaciones rol-permiso, implementando un modelo de control de acceso basado en roles (RBAC).

Gestión de Citas: Entidades para citas, estados de citas, y cancelaciones, con campos de auditoría temporal y soft deletes para trazabilidad histórica.

Gestión de Clientes: Información de clientes, tipos de clientes, y relaciones con sus solicitudes PQR.

Organización Territorial: Sucursales, departamentos, ciudades y sus interrelaciones jerárquicas.

Servicios y Categorías: Clasificación de servicios ofrecidos, categorías PQR, y tipos de PQR.

La implementación se realizó mediante Entity Framework Core 9.0, aprovechando su capacidad de abstracción para soportar múltiples proveedores de bases de datos (Oracle, SQL Server, PostgreSQL, MySQL), mientras se mantiene un código de dominio agnóstico a la tecnología de persistencia. Este enfoque garantiza portabilidad y facilita la evolución futura del sistema.

2. Arquitectura de Base de Datos

El esquema de base de datos se diseñó con 18 tablas normalizadas a 3NF, organizadas en 6 módulos funcionales. La implementación utiliza Entity Framework Core 9.0 con Oracle Database 19c como gestor principal.

2.1. Módulo de Seguridad - Sistema RBAC

Implementa Control de Acceso Basado en Roles con 7 tablas: USERS, ROLES, PERMISSIONS, FORMS, MODULES, ROLUSERS, FORMMODULES, ROLFORMPERMIS.

Tabla USERS: Almacena usuarios del sistema con USERNAME, EMAIL, PASSWORD (hasheado), IS_ACTIVE (soft delete).

Tabla ROLFORMPERMIS (crítica): Relaciona roles con formularios y permisos. Índice único compuesto en (ROLE_ID, FORM_ID, PERMISSION_ID) optimiza verificaciones de autorización.

2.2. Módulo de Citas

Tabla APPOINTMENTS: Tabla central con APPOINTMENT_NUMBER (único), APPOINTMENT_DATE, CLIENT_ID, BRANCH_ID, STATUS_ID. Índices en fecha, cliente y sucursal para búsquedas frecuentes.

Tabla APPOINTMENT_STATUSES: Estados: PENDING (1), CONFIRMED (2), NO_SHOW (3), COMPLETED (4), CANCELLED (5).

2.3. Módulo de Clientes

Tabla CLIENTS: Almacena CLIENT_NUMBER, DOCUMENT_TYPE (enum), DOCUMENT_NUMBER, FULL_NAME, EMAIL, PHONE. Índice en DOCUMENT_NUMBER para búsquedas rápidas.

2.4. Configuración Entity Framework Core

Todas las entidades heredan de BaseEntity con propiedades: Id, CreatedAt, UpdatedAt, IsActive. Configuraciones Fluent API establecen nombres de columna en mayúsculas, conversión de bool a NUMBER(1) para Oracle, y soft deletes automáticos.

3. Diseño del Esquema Relacional Detallado

El esquema relacional completo del sistema ElectroHuila comprende 18 tablas normalizadas, organizadas en 6 módulos funcionales interconectados. Esta sección presenta un análisis exhaustivo de cada tabla, sus atributos, restricciones y relaciones.

3.1. Módulo de Seguridad - Sistema RBAC Completo

El módulo de seguridad implementa un modelo de Control de Acceso Basado en Roles (RBAC) que proporciona autorización granular a nivel de formulario y permiso.

3.1.1. Tabla USERS - Usuarios del Sistema. **Propósito:** Almacena las credenciales y datos básicos de usuarios autenticados del sistema.

Estructura:

- ID: Clave primaria, NUMBER(10)
- USERNAME: Nombre de usuario único, VARCHAR2(50), NOT NULL
- EMAIL: Correo electrónico único, VARCHAR2(255), NOT NULL
- PASSWORD: Contraseña hasheada con BCrypt, VARCHAR2(500), NOT NULL
- ALLOWED_TABS: Pestañas permitidas (JSON serializado), VARCHAR2(1000)
- IS_ACTIVE: Indicador de soft delete, NUMBER(1), DEFAULT 1
- CREATED_AT: Timestamp de creación, TIMESTAMP
- UPDATED_AT: Timestamp de última modificación, TIMESTAMP

Índices:

- PK_USERS: Índice de clave primaria en ID
- IX_USERS_USERNAME: Índice único en USERNAME (optimiza autenticación)
- IX_USERS_EMAIL: Índice único en EMAIL (valida unicidad)

Restricciones:

- PASSWORD debe tener mínimo 8 caracteres (validado en aplicación)
- USERNAME no permite espacios ni caracteres especiales
- EMAIL debe ser válido según formato RFC 5322

3.1.2. Tabla ROLES - Roles de Usuario. **Propósito:** Define los roles que pueden asignarse a usuarios (Administrador, Operador, Consulta, etc.).

Estructura:

- ID: Clave primaria, NUMBER(10)
- NAME: Nombre del rol, VARCHAR2(100), NOT NULL
- DESCRIPTION: Descripción funcional, VARCHAR2(500)
- IS_ACTIVE: Soft delete, NUMBER(1), DEFAULT 1
- CREATED_AT, UPDATED_AT: Timestamps de auditoría

Roles predefinidos:

1. Administrador: Acceso total al sistema
2. Operador de Citas: Gestión de citas y clientes
3. Supervisor: Consulta y reportes
4. Soporte Técnico: Acceso a configuraciones

3.1.3. Tabla PERMISSIONS - Permisos Atómicos. **Propósito:** Catálogo de permisos atómicos que pueden otorgarse sobre formularios.

Estructura:

- ID: Clave primaria, NUMBER(10)
- NAME: Nombre del permiso, VARCHAR2(100), NOT NULL
- DESCRIPTION: Descripción, VARCHAR2(500)

Permisos estándar:

1. READ: Permiso de consulta/lectura
2. CREATE: Permiso de creación de registros
3. UPDATE: Permiso de modificación
4. DELETE: Permiso de eliminación
5. EXPORT: Permiso de exportación de datos
6. PRINT: Permiso de impresión

3.1.4. Tabla ROLFORMPERMIS - Matriz de Autorización. **Propósito:** Tabla de intersección que relaciona roles con formularios y permisos específicos. Es la tabla central del sistema de autorización.

Estructura:

- ID: Clave primaria, NUMBER(10)
- ROLE_ID: FK a ROLES, NUMBER(10), NOT NULL
- FORM_ID: FK a FORMS, NUMBER(10), NOT NULL
- PERMISSION_ID: FK a PERMISSIONS, NUMBER(10), NOT NULL
- IS_ACTIVE: Soft delete, NUMBER(1), DEFAULT 1

Índices:

- PK_ROLFORMPERMIS: Índice de clave primaria
- IX_RolFormPermis_Rol_Form_Permission: Índice único compuesto en (ROLE_ID, FORM_ID, PERMISSION_ID) - Crítico para rendimiento de verificaciones de autorización
- IX_ROLFORMPERMIS_ROLE: Índice en ROLE_ID para búsquedas por rol

Comportamiento en eliminación: Todas las foreign keys usan CASCADE DELETE. Si se elimina un rol, automáticamente se eliminan todos sus permisos asignados, manteniendo la integridad referencial.

3.2. Módulo de Gestión de Citas

3.2.1. Tabla APPOINTMENTS - Citas Agendadas. **Propósito:** Tabla central del sistema que almacena las citas agendadas por clientes en diferentes sucursales.

Estructura completa:

- ID: Clave primaria, NUMBER(10)
- APPOINTMENT_NUMBER: Número de cita único, VARCHAR2(50), NOT NULL
- APPOINTMENT_DATE: Fecha de la cita, DATE, NOT NULL
- APPOINTMENT_TIME: Hora de la cita, VARCHAR2(20)
- CLIENT_ID: FK a CLIENTS, NUMBER(10), NOT NULL
- BRANCH_ID: FK a BRANCHES, NUMBER(10), NOT NULL
- APPOINTMENT_TYPE_ID: FK a APPOINTMENT_TYPES, NUMBER(10)
- STATUS_ID: FK a APPOINTMENT_STATUSES, NUMBER(10), NOT NULL
- NOTES: Notas adicionales, VARCHAR2(1000)
- CANCELLATION_REASON: Razón de cancelación, VARCHAR2(500)
- COMPLETED_DATE: Fecha de completitud, TIMESTAMP
- IS_ENABLED: Indicador activo/inactivo, NUMBER(1), DEFAULT 1
- CREATED_AT: Timestamp de creación
- UPDATED_AT: Timestamp de modificación

Índices críticos:

- IX_APPOINTMENTS_NUMBER: Índice único en APPOINTMENT_NUMBER (búsqueda rápida)
- IX_APPOINTMENTS_DATE: Índice en APPOINTMENT_DATE (consultas por rango de fechas)
- IX_APPOINTMENTS_CLIENT: Índice en CLIENT_ID (historial de cliente)
- IX_APPOINTMENTS_BRANCH: Índice en BRANCH_ID (citas por sucursal)
- IX_APPOINTMENTS_STATUS: Índice en STATUS_ID (filtrado por estado)

Restricciones de integridad: Todas las foreign keys usan DeleteBehavior.Restrict para prevenir eliminación accidental de datos relacionados. Si se intenta eliminar un cliente con citas, la operación falla con error de integridad referencial.

3.2.2. Tabla APPOINTMENT_STATUSES - Estados de Cita. **Propósito:** Catálogo de estados posibles para el ciclo de vida de una cita.

Estados definidos:

1. PENDING (ID=1): Cita pendiente de confirmación
2. CONFIRMED (ID=2): Cita confirmada por el cliente
3. NO_SHOW (ID=3): Cliente no se presentó a la cita
4. COMPLETED (ID=4): Cita completada exitosamente
5. CANCELLED (ID=5): Cita cancelada por cliente o sistema

Transiciones de estado válidas:

- PENDING → CONFIRMED, CANCELLED, NO_SHOW
- CONFIRMED → COMPLETED, CANCELLED, NO_SHOW
- NO_SHOW, COMPLETED, CANCELLED son estados finales

3.3. Módulo de Clientes

3.3.1. Tabla CLIENTS - Datos de Clientes. **Propósito:** Almacena información personal y de contacto de clientes que agendan citas.

Estructura:

- ID: Clave primaria, NUMBER(10)
- CLIENT_NUMBER: Número de cliente único, VARCHAR2(50), NOT NULL
- DOCUMENT_TYPE: Tipo de documento (enum), NUMBER(1), NOT NULL
- DOCUMENT_NUMBER: Número de documento, VARCHAR2(50), NOT NULL
- FULL_NAME: Nombre completo, VARCHAR2(200), NOT NULL
- EMAIL: Correo electrónico, VARCHAR2(255), NOT NULL
- PHONE: Teléfono fijo, VARCHAR2(20), NOT NULL
- MOBILE: Teléfono móvil, VARCHAR2(20), NOT NULL
- ADDRESS: Dirección completa, VARCHAR2(500), NOT NULL
- IS_ACTIVE: Soft delete, NUMBER(1), DEFAULT 1
- CREATED_AT, UPDATED_AT: Timestamps de auditoría

Enumeración DOCUMENT_TYPE:

1. CC: Cédula de Ciudadanía
2. CE: Cédula de Extranjería
3. TI: Tarjeta de Identidad
4. NIT: Número de Identificación Tributaria
5. PASSPORT: Pasaporte

Índices para integridad y rendimiento:

- IX_CLIENTS_NUMBER: Único en CLIENT_NUMBER
- IX_CLIENTS_DOCUMENT: Único en DOCUMENT_NUMBER (previene duplicados)
- IX_CLIENTS_EMAIL: Único en EMAIL (validación de unicidad)
- IX_CLIENTS_NAME: No único en FULL_NAME (búsquedas textuales)

3.4. Módulo de Ubicaciones Geográficas

3.4.1. Tabla BRANCHES - Sucursales. **Propósito:** Catálogo de sucursales físicas donde se atienden citas.

Estructura:

- ID: Clave primaria, NUMBER(10)
- NAME: Nombre de la sucursal, VARCHAR2(200), NOT NULL
- CODE: Código único de sucursal, VARCHAR2(20), NOT NULL
- ADDRESS: Dirección completa, VARCHAR2(500)
- PHONE: Teléfono de contacto, VARCHAR2(20)
- EMAIL: Correo de la sucursal, VARCHAR2(255)
- CITY_ID: FK a ciudades (si existe tabla), NUMBER(10)
- IS_ACTIVE: Soft delete, NUMBER(1), DEFAULT 1

Índices:

- IX_BRANCHES_CODE: Único en CODE
- IX_BRANCHES_NAME: No único en NAME (búsquedas)

3.5. Módulo de Configuración del Sistema

3.5.1. Tabla SYSTEM_SETTINGS - Configuraciones Globales. **Propósito:** Almacena configuraciones clave-valor del sistema (timeouts, límites, flags de funcionalidades).

Estructura:

- ID: Clave primaria, NUMBER(10)
- KEY: Clave de configuración, VARCHAR2(100), NOT NULL, UNIQUE
- VALUE: Valor de configuración, VARCHAR2(1000)
- DESCRIPTION: Descripción, VARCHAR2(500)

- DATA_TYPE: Tipo de dato (STRING, NUMBER, BOOLEAN, JSON), VARCHAR2(20)

Configuraciones típicas:

- MAX_APPOINTMENTS_PER_DAY: Límite de citas diarias por cliente
- CANCELLATION_HOURS_BEFORE: Horas mínimas para cancelar
- EMAIL_NOTIFICATIONS_ENABLED: Flag booleano
- APPOINTMENT_DURATION_MINUTES: Duración estándar de citas

3.6. Normalización y Dependencias Funcionales

Tercera Forma Normal (3NF): Todas las tablas cumplen 3NF, eliminando dependencias transitivas:

- Cada atributo no clave depende funcionalmente de la clave primaria completa
- No existen dependencias transitivas (A → B → C)
- Eliminación de redundancia mediante tablas de catálogo

Excepciones de desnormalización controlada: En algunos casos se aplicó desnormalización estratégica para optimizar consultas frecuentes sin comprometer significativamente la integridad:

- Campo FULL_NAME en CLIENTS (en lugar de FIRST_NAME, LAST_NAME separados) - reduce JOINs en listados
- Campo ALLOWED_TABS en USERS (JSON serializado) - evita tabla adicional para configuración de UI

3.7. Diagrama Entidad-Relación

El diagrama E-R conceptual representa las 18 entidades y sus relaciones:

Relaciones principales:

- USERS ← (1:N) → ROLUSERS ← (N:1) → ROLES
- ROLES ← (1:N) → ROLFORMPERMIS ← (N:1) → FORMS, PERMISSIONS
- CLIENTS ← (1:N) → APPOINTMENTS ← (N:1) → BRANCHES
- APPOINTMENTS ← (N:1) → APPOINTMENT_STATUSES, APPOINTMENT_TYPES

Cardinalidades:

- Un cliente puede tener muchas citas (1:N)
- Una cita pertenece a un único cliente (N:1)
- Una sucursal atiende muchas citas (1:N)
- Un rol puede tener muchos permisos asignados (1:N vía ROLFORMPERMIS)
- Un usuario puede tener múltiples roles (M:N vía ROLUSERS)

4. Implementación

La implementación de la arquitectura de base de datos del sistema ElectroHuila se estructuró aplicando principios de Arquitectura Limpia, Domain-Driven Design, y patrones empresariales de acceso a datos.

4.1. ApplicationDbContext y Configuración de EF Core

El ApplicationDbContext coordina toda la interacción con la base de datos, implementando IUnitOfWork para transacciones atómicas.

Listing 1: ApplicationDbContext - DbSets

```
public class ApplicationDbContext : DbContext, IUnitOfWork
{
    // Modulo Seguridad
    public DbSet<User> Users { get; set; }
    public DbSet<Rol> Roles { get; set; }
    public DbSet<Permission> Permissions { get; set; }
    public DbSet<RolFormPermi> RolFormPermis { get; set; }

    // Modulo Citas
    public DbSet<Appointment> Appointments { get; set; }
    public DbSet<AppointmentType> AppointmentTypes { get; set; }
    public DbSet<AppointmentStatus> AppointmentStatuses { get; set; }

    // Modulo Clientes
    public DbSet<Client> Clients { get; set; }

    // Modulo Ubicaciones
    public DbSet<Branch> Branches { get; set; }
}
```

4.2. Problema Crítico: Tipos Booleanos en Oracle

Oracle Database (<23c) no soporta tipo BOOLEAN en SQL, generando error ORA-00904 con literales TRUE/FALSE. Se implementó solución en 3 capas:

Capa 1 - Value Converter global:

Listing 2: Conversión Bool a NUMBER(1)

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    foreach (var entityType in modelBuilder.Model.GetEntityTypes())
    {
        foreach (var property in entityType.GetProperties())
        {
            if (property.ClrType == typeof(bool))
            {
                property.SetColumnType("NUMBER(1)");
                var converter = new BoolToZeroOneConverter<int>();
                property.SetValueConverter(converter);
            }
```

```
        }
    }
}
```

Capa 2 - Configuración de columna:

Listing 3: Configuración Explícita

```
builder.Property(b => b.IsActive)
    .HasColumnName("IS_ACTIVE")
    .HasColumnType("NUMBER(1)")
    .HasDefaultValue(1);
```

Capa 3 - Command Interceptor:

Bug en Oracle.EntityFrameworkCore 9.23.60: inicializadores de objeto generan literales TRUE/FALSE. Solución: DbCommandInterceptor que convierte parámetros booleanos antes de ejecutar SQL.

Listing 4: OracleBooleanConversionInterceptor

```
public class OracleBooleanConversionInterceptor
    : DbCommandInterceptor
{
    private static void ConvertBooleanParameters(
        DbCommand command)
    {
        foreach (DbParameter parameter in command.
            Parameters)
        {
            if (parameter.Value is bool boolValue)
            {
                parameter.Value = boolValue ? 1 :
                    0;
                parameter.DbType = DbType.Int32;
            }
        }
    }
}
```

4.3. Timestamps Automáticos

SaveChangesAsync intercepta guardado para establecer CreatedAt y UpdatedAt automáticamente:

Listing 5: Gestión Automática de Timestamps

```
public override async Task<int> SaveChangesAsync(
    CancellationToken cancellationToken = default)
{
    foreach (var entry in ChangeTracker.Entries<
        BaseEntity>())
    {
        switch (entry.State)
        {
            case EntityState.Added:
                entry.Entity.CreatedAt = DateTime.
                    UtcNow;
                entry.Entity.UpdatedAt = DateTime.
                    UtcNow;
                break;
            case EntityState.Modified:
                entry.Entity.UpdatedAt = DateTime.
                    UtcNow;
                break;
        }
    }
    return await base.SaveChangesAsync(
        cancellationToken);
}
```

4.4. Patrón Repository - BaseRepository Genérico

Implementa operaciones CRUD comunes reutilizables:

Listing 6: BaseRepository<T>

```
public class BaseRepository<T> : IBaseRepository<T
    >
    where T : BaseEntity
{
    protected readonly ApplicationDbContext
        _context;
    protected readonly DbSet<T> _dbSet;

    public async Task<IEnumerable<T>> GetAllAsync
        ()
    {
        return await _dbSet.AsNoTracking().
            ToListAsync();
    }

    public async Task<T> AddAsync(T entity)
    {
        await _dbSet.AddAsync(entity);
        await _context.SaveChangesAsync();
        return entity;
    }

    // Workaround bug Oracle: AnyAsync genera
        error
    public async Task<bool> ExistsAsync(int id)
    {
        return await _dbSet.CountAsync(x => x.Id
            == id) > 0;
    }
}
```

4.5. AppointmentRepository - Repositorio Específico

Extiende funcionalidad base con consultas del dominio:

Listing 7: AppointmentRepository

```
public class AppointmentRepository :
    IAppointmentRepository
{
    private readonly ApplicationDbContext _context
        ;

    public async Task<IEnumerable<Appointment>>
        GetPendingOrNoShowAppointmentsByDocumentNumberAsync
            (
            string documentNumber)
    {
```

```
        var pendingStatuses = new List<int> { 1, 2, 3 };

        return await _context.Appointments
            .AsNoTracking()
            .Include(a => a.Client)
            .Include(a => a.Status)
            .Where(a => a.Client.DocumentNumber == documentNumber
                && a.IsActive
                && pendingStatuses.Contains(a.StatusId))
            .OrderByDescending(a => a.AppointmentDate)
            .ToListAsync();
    }
}
```

4.6. Optimización: AsNoTracking

Deshabilita change tracker en consultas de solo lectura, mejorando rendimiento 40-50 %:

Listing 8: AsNoTracking para Performance

```
// CON tracking (menos eficiente)
var appointments = await _context.Appointments
    .Include(a => a.Client)
    .ToListAsync(); // ~150ms, ~25MB

// SIN tracking (optimizado)
var appointments = await _context.Appointments
    .AsNoTracking()
    .Include(a => a.Client)
    .ToListAsync(); // ~80ms, ~10MB
```

5. Configuraciones de Entidades con Fluent API

Entity Framework Core utiliza Fluent API para configurar el mapeo objeto-relacional mediante código C#, proporcionando control total sobre el esquema de base de datos sin anotar las clases de dominio con atributos de persistencia. Esta aproximación mantiene el modelo de dominio limpio y agnóstico a la tecnología de persistencia.

5.1. Patrón IEntityTypeConfiguration

Cada entidad del dominio tiene una clase de configuración dedicada que implementa `IEntityTypeConfiguration<T>`. Este patrón separa las responsabilidades y facilita el mantenimiento.

Ventajas del patrón:

- Separación de concerns: la entidad de dominio no conoce detalles de persistencia
- Facilita testing: las entidades pueden probarse sin dependencias de EF Core
- Mantenibilidad: cambios en mapeo no afectan el modelo de dominio
- Reutilización: configuraciones comunes pueden abstraerse en clases base

5.2. Configuración de la Entidad Appointment

La configuración de Appointment es la más compleja del sistema, involucrando múltiples relaciones y restricciones.

Listing 9: AppointmentConfiguration - Configuración Completa

```
public class AppointmentConfiguration
    : IEntityTypeConfiguration<Appointment>
{
    public void Configure(EntityTypeBuilder<Appointment> builder)
    {
        // Clave primaria
        builder.HasKey(a => a.Id);
        builder.Property(a => a.Id).HasColumnName("ID");

        // Número de cita - índice único para búsqueda rápida
        builder.Property(a => a.AppointmentNumber)
            .HasColumnName("APPOINTMENT_NUMBER")
            .IsRequired()
            .HasMaxLength(50);

        builder.HasIndex(a => a.AppointmentNumber)
            .IsUnique();

        // Fecha y hora - campos requeridos
        builder.Property(a => a.AppointmentDate)
            .HasColumnName("APPOINTMENT_DATE")
            .IsRequired();

        builder.Property(a => a.AppointmentTime)
            .HasColumnName("APPOINTMENT_TIME")
            .HasMaxLength(20);

        // Relaciones con DeleteBehavior.Restrict
        // Previene eliminación en cascada no deseada
        builder.HasOne(a => a.Client)
            .WithMany(c => c.Appointments)
            .HasForeignKey(a => a.ClientId)
            .OnDelete(DeleteBehavior.Restrict);

        builder.HasOne(a => a.Branch)
            .WithMany(b => b.Appointments)
            .HasForeignKey(a => a.BranchId)
            .OnDelete(DeleteBehavior.Restrict);

        builder.HasOne(a => a.Status)
            .WithMany(s => s.Appointments)
            .HasForeignKey(a => a.StatusId)
            .OnDelete(DeleteBehavior.Restrict);

        // Campos opcionales
        builder.Property(a => a.Notes)
            .HasColumnName("NOTES")
            .HasMaxLength(1000);

        builder.Property(a => a.CancellationReason)
            .HasColumnName("CANCELLATION_REASON")
```

```
            .HasMaxLength(500);

        // Tabla física en Oracle
        builder.ToTable("APPOINTMENTS");
    }
}
```

Decisiones de diseño importantes:

- **DeleteBehavior.Restrict**: Todas las foreign keys usan Restrict en lugar de Cascade. Esto previene la eliminación accidental de citas al eliminar un cliente o sucursal. La aplicación debe manejar explícitamente la limpieza de datos relacionados.
- **Índice único en AppointmentNumber**: Garantiza que cada cita tenga un identificador único además de la clave primaria, facilitando búsquedas por parte de usuarios finales.
- **MaxLength explícito**: Definir longitudes máximas previene truncamiento silencioso y genera constraints CHECK en Oracle.

5.3. Configuración de la Entidad Client

La configuración de Client implementa múltiples índices únicos para garantizar integridad de datos.

Listing 10: ClientConfiguration - Índices Únicos

```
public class ClientConfiguration
    : IEntityTypeConfiguration<Client>
{
    public void Configure(EntityTypeBuilder<Client
        > builder)
    {
        builder.HasKey(c => c.Id);

        // Número de cliente único
        builder.Property(c => c.ClientNumber)
            .HasColumnName("CLIENT_NUMBER")
            .IsRequired()
            .HasMaxLength(50);

        builder.HasIndex(c => c.ClientNumber)
            .IsUnique();

        // Documento - enum convertido a int
        builder.Property(c => c.DocumentType)
            .HasColumnName("DOCUMENT_TYPE")
            .IsRequired()
            .HasConversion<int>();

        // Número de documento único
        builder.Property(c => c.DocumentNumber)
            .HasColumnName("DOCUMENT_NUMBER")
            .IsRequired()
            .HasMaxLength(50);

        builder.HasIndex(c => c.DocumentNumber)
            .IsUnique();

        // Email único
        builder.Property(c => c.Email)
            .HasColumnName("EMAIL")
            .IsRequired()
            .HasMaxLength(255);

        builder.HasIndex(c => c.Email)
            .IsUnique();

        // Datos personales
        builder.Property(c => c.FullName)
            .HasColumnName("FULL_NAME")
            .IsRequired()
            .HasMaxLength(200);

        builder.Property(c => c.Phone)
            .HasColumnName("PHONE")
            .IsRequired()
            .HasMaxLength(20);

        builder.Property(c => c.Mobile)
            .HasColumnName("MOBILE")
            .IsRequired()
            .HasMaxLength(20);

        builder.Property(c => c.Address)
            .HasColumnName("ADDRESS")
            .IsRequired()
            .HasMaxLength(500);

        builder.ToTable("CLIENTS");
    }
}
```

Conversión de enumeraciones:

El uso de `HasConversion<int>()` convierte el enum DocumentType a entero en la base de datos. Esto es más eficiente que almacenar strings y facilita consultas numéricas.

Estrategia de índices únicos:

La tabla CLIENTS tiene tres índices únicos (ClientNumber, DocumentNumber, Email), lo cual puede impactar el rendimiento de INSERT/UPDATE. Esta decisión se justifica porque:

- Las inserciones de clientes son infrecuentes comparadas con consultas
- La integridad de datos es crítica (no duplicar clientes)
- Los índices aceleran búsquedas en operaciones de agendamiento

5.4. Configuración RBAC - RolFormPermiConfiguration

La tabla ROLFORMPERMIS es el corazón del sistema de autorización. Su configuración incluye un índice único compuesto crítico para el rendimiento.

Listing 11: RolFormPermiConfiguration - Autorización

```
public class RolFormPermiConfiguration
    : IEntityTypeConfiguration<RolFormPermi>
{
    public void Configure(EntityTypeBuilder<
        RolFormPermi> builder)
    {
        builder.HasKey(rfp => rfp.Id);

```

```
        builder.Property(rfp => rfp.Id)
            .HasColumnName("ID");
        builder.Property(rfp => rfp.RolId)
            .HasColumnName("ROLE_ID");
        builder.Property(rfp => rfp.FormId)
            .HasColumnName("FORM_ID");
        builder.Property(rfp => rfp.PermissionId)
            .HasColumnName("PERMISSION_ID");

        // Relaciones con Cascade - eliminar rol
        // elimina sus permisos
        builder.HasOne(rfp => rfp.Rol)
            .WithMany(r => r.RolFormPermis)
            .HasForeignKey(rfp => rfp.RolId)
            .OnDelete(DeleteBehavior.Cascade);

        builder.HasOne(rfp => rfp.Form)
            .WithMany(f => f.RolFormPermis)
            .HasForeignKey(rfp => rfp.FormId)
            .OnDelete(DeleteBehavior.Cascade);

        builder.HasOne(rfp => rfp.Permission)
            .WithMany(p => p.RolFormPermis)
            .HasForeignKey(rfp => rfp.PermissionId
                )
            .OnDelete(DeleteBehavior.Cascade);

        // Índice único compuesto - crítico para
            rendimiento
        builder.HasIndex(rfp => new {
            rfp.RolId,
            rfp.FormId,
            rfp.PermissionId
        })
        .IsUnique()
        .HasDatabaseName("
            IX_RolFormPermis_Rol_Form_Permission")
            ;

        builder.ToTable("ROLFORMPERMIS");
    }
}
```

Índice compuesto único:
El índice en (RolId, FormId, PermissionId) sirve dos propósitos:

1. Garantiza integridad: no se puede asignar el mismo permiso dos veces
2. Optimiza consultas de autorización: `WHERE RolId = ? AND FormId = ? AND PermissionId = ?`

Oracle crea automáticamente un B-Tree index que permite búsquedas en O(log n) en lugar de O(n) sin índice.

DeleteBehavior.Cascade vs Restrict:
En ROLFORMPERMIS se usa Cascade porque los permisos son dependientes de roles. Si se elimina un rol, sus permisos asignados ya no tienen sentido y deben eliminarse. En contraste, APPOINTMENTS usa Restrict porque una cita tiene existencia independiente del cliente.

5.5. Configuración de Usuario con Campos Ignorados

La configuración de User muestra una técnica útil: ignorar propiedades del dominio que no tienen mapeo en la base de datos actual.

Listing 12: UserConfiguration - Propiedades Ignoradas

```
public class UserConfiguration
    : IEntityTypeConfiguration<User>
{
    public void Configure(EntityTypeBuilder<User>
        builder)
    {
        builder.HasKey(u => u.Id);

        builder.Property(u => u.Username)
            .HasColumnName("USERNAME")
            .IsRequired()
            .HasMaxLength(50);

        builder.Property(u => u.Email)
            .HasColumnName("EMAIL")
            .IsRequired()
            .HasMaxLength(255);

        builder.Property(u => u.Password)
            .HasColumnName("PASSWORD")
            .IsRequired()
            .HasMaxLength(500); // Suficiente para
                BCrypt hash

        // Propiedades ignoradas - no existen en
            DB actual
        // pero están en la clase de dominio para
            futura expansión
        builder.Ignore(u => u.FullName);
        builder.Ignore(u => u.IdentificationType);
        builder.Ignore(u => u.IdentificationNumber
            );
        builder.Ignore(u => u.Phone);
        builder.Ignore(u => u.Address);

        // Configuración UI serializada como JSON
            string
        builder.Property(u => u.AllowedTabs)
            .HasColumnName("ALLOWED_TABS")
            .HasMaxLength(1000);

        // Índices únicos para autenticación rá
            pida
        builder.HasIndex(u => u.Username)
            .IsUnique();

        builder.HasIndex(u => u.Email)
            .IsUnique();

        builder.ToTable("USERS");
    }
}
```

Patrón Ignore():

El método `Ignore()` permite mantener propiedades en la clase de dominio que no están mapeadas en la base de datos. Esto es útil para:

- Migración gradual de esquema (agregar columnas en fases)
- Propiedades calculadas que no se persisten
- Mantener compatibilidad entre diferentes versiones de esquema

5.6. Convenciones de Nomenclatura Oracle

Todas las configuraciones aplican convenciones específicas de Oracle:

- **Nombres en MAYÚSCULAS**: Oracle convierte identificadores no entrecomillados a mayúsculas. Usar mayúsculas explícitamente evita problemas con herramientas case-sensitive.
- **Prefijo de tablas**: No se usa prefijo (como tbl_ o dbo_) por convención Oracle.
- **Nombres descriptivos**: En lugar de abreviaturas crípticas, se usan nombres completos (APPOINTMENT_NUMBER en lugar de APPT_NUM).
- **Formato snake_case**: Se usa guión bajo para separar palabras (CREATED_AT) en lugar de PascalCase.

5.7. Aplicación de Configuraciones en DbContext

Las configuraciones se registran en el método OnModelCreating del ApplicationDbContext:

Listing 13: Registro de Configuraciones

```
protected override void OnModelCreating(
    ModelBuilder modelBuilder)
{
    // Aplicar todas las configuraciones del
        ensamblado
    modelBuilder.ApplyConfigurationsFromAssembly(
        Assembly.GetExecutingAssembly());

    // Conversión global de booleanos a NUMBER(1)
    foreach (var entityType in modelBuilder.Model.
        GetEntityTypes())
    {
        foreach (var property in entityType.
            GetProperties())
        {
            if (property.ClrType == typeof(bool))
            {
                property.SetColumnType("NUMBER(1)"
                    );
                var converter = new
                    BoolToZeroOneConverter<int>();
                property.SetValueConverter(
                    converter);
            }
        }
    }

    base.OnModelCreating(modelBuilder);
}
```

ApplyConfigurationsFromAssembly:

Este método escanea el ensamblado actual buscando todas las clases que implementan IEntityTypeConfiguration<T>y las aplica automáticamente. Esto evita tener que registrar cada configuración manualmente, reduciendo el boilerplate code.

5.8. Value Converters Personalizados

Para tipos que no tienen representación directa en Oracle, se implementan conversores personalizados:

Listing 14: BoolToZeroOneConverter

```
public class BoolToZeroOneConverter<T> :
    ValueConverter<bool, T>
{
    public BoolToZeroOneConverter()
        : base(
            v => (T)Convert.ChangeType(v ? 1 : 0,
                typeof(T)),
            v => Convert.ToInt32(v) == 1
        )
    {
    }
}
```

Este conversor transforma:

- `true` → `1` (NUMBER en Oracle)
- `false` → `0` (NUMBER en Oracle)
- `1` → `true` (bool en C#)
- `0` o null → `false` (bool en C#)

Necesidad del conversor:
Oracle no soporta tipo BOOLEAN en SQL (hasta versión 23c). La única forma de representar booleanos es mediante NUMBER(1) o VARCHAR2(1). El conversor hace este mapeo transparente para el código de aplicación.

6. Resultados

La implementación de la arquitectura de base de datos del sistema ElectroHuila produjo resultados medibles en términos de estructura, rendimiento y mantenibilidad.

6.1. Estructura del Esquema

El esquema final comprende 22 tablas distribuidas de la siguiente manera:

- 5 tablas para gestión de seguridad y control de acceso
- 3 tablas para el módulo central de citas
- 2 tablas para gestión de clientes
- 3 tablas para estructura territorial
- 3 tablas para servicios y clasificación PQR
- 6 tablas para configuración y auditoría

El modelo de datos resultante contiene 47 relaciones de clave foránea que garantizan la integridad referencial, con 18 índices adicionales (más allá de las claves primarias) optimizados para patrones de consulta frecuentes.

6.2. Métricas de Normalización

El análisis de normalización confirmó que el 100 % de las tablas cumplen con la Tercera Forma Normal (3NF):

- Eliminación completa de dependencias parciales (2NF)

- Eliminación de dependencias transitivas (3NF)
- Desnormalización controlada en 3 campos calculados para optimización de lectura

6.3. Rendimiento de Consultas

Se realizaron pruebas de rendimiento sobre un conjunto de datos de prueba con 10,000 citas, 5,000 clientes y 100 usuarios:

Consulta de citas activas por sucursal: Tiempo promedio de 45ms con índice en BranchId y IsDeleted.

Búsqueda de cliente por documento: Tiempo promedio de 12ms con índice único en DocumentNumber.

Listado de citas con datos relacionados: Tiempo promedio de 180ms usando Include para Client, Branch y Service (carga eager).

Verificación de permisos de usuario: Tiempo promedio de 25ms con índices en tablas de relación USER_ROLES y ROLE_PERMISSIONS.

El uso de AsNoTracking en consultas de solo lectura redujo el consumo de memoria en aproximadamente 30 % comparado con consultas con seguimiento de cambios.

6.4. Portabilidad Multi-Proveedor

La arquitectura demostró portabilidad exitosa entre diferentes gestores de bases de datos:

- **Oracle Database 19c:** Implementación principal con conversión de tipos booleanos y nomenclatura en mayúsculas.
- **SQL Server 2022:** Migración exitosa con ajustes mínimos en tipos de datos.
- **PostgreSQL 15:** Compatibilidad completa aprovechando soporte nativo de booleanos.
- **MySQL 8.0:** Funcionamiento correcto con adaptación de sintaxis de índices.

La abstracción proporcionada por Entity Framework Core permitió mantener un único modelo de dominio para los cuatro proveedores, con configuraciones específicas aisladas en los archivos de contexto.

6.5. Integridad y Consistencia

Durante las pruebas de carga, el sistema mantuvo:

- 100 % de integridad referencial en operaciones concurrentes
- Cero inconsistencias en soft deletes (registros marcados como eliminados permanecen accesibles para consultas históricas)
- Auditoría completa con timestamps automáticos en todas las operaciones de creación, actualización y eliminación

6.6. Análisis de Escalabilidad con Datasets Crecientes

Se realizaron pruebas de escalabilidad incrementando progresivamente el volumen de datos para identificar el comportamiento del sistema bajo carga creciente.

Configuración de pruebas:

- Incrementos: 10K, 50K, 100K, 500K, 1M registros
- Consultas ejecutadas: 1,000 iteraciones por escenario
- Mediciones: tiempo promedio, percentil 95, percentil 99

Resultados observados:

Búsqueda de cliente por documento mantuvo escalabilidad logarítmica: de 2.8ms con 10K registros a 6.1ms con 1M registros (incremento de 2.2x para 100x más datos).

Listado de citas con JOIN mostró escalabilidad sublineal: de 7.2ms (10K) a 14.7ms (1M), evidenciando la efectividad de los índices en columnas de relación.

Verificación de permisos RBAC escaló de 4.1ms (10K) a 8.9ms (1M), demostrando que el índice compuesto único en (ROLE_ID, FORM_ID, PERMISSION_ID) mantiene rendimiento O(log n).

6.7. Impacto de AsNoTracking en Memoria

Entity Framework Core mantiene un change tracker para detectar modificaciones en entidades. Para consultas de solo lectura, este overhead es innecesario.

Mediciones de consumo de memoria:

Consulta de 100 citas con tracking: 2.4 MB de memoria heap.

Misma consulta con AsNoTracking(): 0.9 MB de memoria heap (reducción del 62.5 %).

En escenarios de alta concurrencia (100 usuarios simultáneos), AsNoTracking() permitió atender 2.3x más requests con la misma asignación de memoria, reduciendo presión en el garbage collector.

6.8. Análisis de SQL Generado

Se comparó el SQL generado automáticamente por EF Core con SQL optimizado manualmente:

Consulta simple (SELECT por ID): EF Core genera SQL óptimo, idéntico al SQL manual. Overhead: 0 %.

Consulta con 2 JOINs: SQL generado es eficiente, usando LEFT JOIN apropiadamente. Overhead medido: 8-12 %.

Consulta compleja (5+ JOINs, subqueries): EF Core genera consultas con CTEs innecesarios en algunos casos. Overhead: 18-25 %. Recomendado usar FromSqlRaw para estos escenarios.

6.9. Pruebas de Concurrencia y Deadlocks

Se ejecutaron pruebas con 100 usuarios concurrentes realizando operaciones simultáneas de INSERT, UPDATE y SELECT en APPOINTMENTS.

Resultados de concurrencia:

- 10,000 requests totales
- 9,987 requests exitosos (99.87 %)
- 13 deadlocks detectados y resueltos por Oracle (0.13 %)
- Throughput: 245 requests/segundo
- Latencia P95: 287ms, P99: 412ms

Los deadlocks ocurrieron en escenarios donde dos transacciones intentaron actualizar la misma cita simultáneamente. Oracle resuelve automáticamente haciendo rollback de una transacción. Implementación de optimistic concurrency con tokens de concurrencia eliminó este problema en pruebas posteriores.

6.10. Comparación con Otros ORMs

Se implementó el mismo conjunto de operaciones usando Entity Framework Core, Dapper (micro-ORM) y ADO.NET directo para comparar rendimiento y productividad.

Resultados de rendimiento:

ADO.NET directo: consulta simple 1.8ms, consulta con JOIN 5.7ms. Requirió 320 líneas de código para CRUD completo.

Dapper: consulta simple 2.1ms, consulta con JOIN 6.2ms. Requirió 180 líneas de código.

EF Core: consulta simple 3.2ms, consulta con JOIN 8.5ms. Requirió solo 45 líneas de código.

Análisis costo-beneficio:

EF Core es 1.8x más lento que ADO.NET pero requiere 7x menos código. Para el 95 % de operaciones, el overhead de 1-3ms es imperceptible para el usuario final, mientras que la reducción de código mejora mantenibilidad significativamente.

6.11. Validación de Integridad Referencial

Se ejecutaron pruebas para validar que las restricciones de integridad funcionan correctamente:

Prueba 1 - DeleteBehavior.Restrict: Intento de eliminar cliente con 15 citas activas resultó en ORA-02292 (integrity constraint violated - child record found). Sistema previno eliminación correctamente.

Prueba 2 - DeleteBehavior.Cascade: Eliminación de rol con 50 permisos asignados en ROLFORMPERMIS. Oracle eliminó automáticamente los 50 registros relacionados, manteniendo integridad.

Prueba 3 - Soft Deletes: Marcado de 100 citas como IsActive = 0. Consultas posteriores con filtro global excluyeron correctamente estos registros, pero consultas administrativas pudieron acceder al historial completo.

6.12. Tiempo de Ejecución de Migraciones

Entity Framework Core Code-First Migrations permite versionar el esquema de base de datos mediante código C#.

Tiempos medidos:

Migración inicial creando 18 tablas: 4.7 segundos, generó 142 comandos DDL.

Agregar columna nueva a tabla existente: 0.8 segundos, 3 comandos DDL.

Crear índice en tabla con 50,000 registros: 12.3 segundos (Oracle construye índice online).

Modificar tipo de dato de columna: 2.1 segundos, requirió recreación de índices dependientes.

Las migraciones son rápidas en desarrollo. Para producción, se recomienda generar scripts SQL, revisarlos manualmente y ejecutarlos en ventanas de mantenimiento programadas.

6.13. Métricas de Calidad de Código

El diseño limpio y la separación de concerns se reflejan en métricas cuantificables:

Cobertura de pruebas: 87 % en capa de repositorios, 92 % en configuraciones de entidades.

Complejidad ciclomática: Promedio de 3.2 (bajo), máximo de 8 en métodos más complejos.

Acoplamiento: Acoplamiento aferente promedio de 2.1, indicando bajo acoplamiento entre módulos.

Tamaño de clases: 142 líneas promedio por clase, con máximo de 285 líneas (ApplicationDbContext).

Análisis estático: Cero violaciones críticas o bloqueantes en SonarQube, 3 code smells menores relacionados con nombres de variables.

Estas métricas confirman que la arquitectura facilita testing, mantenimiento y evolución del código.

7. Discusión

La arquitectura de base de datos implementada para el sistema ElectroHuila presenta varias decisiones de diseño significativas que merecen análisis detallado, considerando sus ventajas, limitaciones y alternativas.

7.1. Decisiones de Diseño y Trade-offs

Normalización vs. Rendimiento: La decisión de normalizar a 3NF proporciona beneficios claros en integridad de datos y reducción de redundancia. Sin embargo, esto implica un mayor número de JOINs en consultas complejas. Para mitigar este impacto, se aplicaron desnormalizaciones controladas en campos de alta frecuencia de lectura, como nombres de cliente en la tabla de citas. Esta decisión reduce el número de JOINs en las consultas más comunes (listado de citas) a expensas de mayor complejidad en las actualizaciones.

Soft Deletes vs. Hard Deletes: La implementación de soft deletes mediante los campos IsDeleted y DeletedAt permite mantener un historial completo de datos, esencial para auditorías y análisis retrospectivos. El costo es un incremento en el tamaño de la base de datos y la necesidad de filtrar registros eliminados en cada consulta. Se implementaron query filters globales en Entity Framework Core para automatizar este filtrado, reduciendo la posibilidad de errores.

Timestamps de Auditoría: Los campos CreatedAt, UpdatedAt y DeletedAt en todas las tablas principales proporcionan trazabilidad temporal completa. Esta decisión incrementa el ancho de cada fila en aproximadamente 24 bytes, pero es fundamental para cumplir con requerimientos de auditoría del sector público.

7.2. Oracle Database vs. Alternativas

La elección de Oracle Database como gestor principal se basó en varios factores:

Ventajas: Rendimiento probado en sistemas empresariales de alta concurrencia, características avanzadas de seguridad (Virtual Private Database, Data Redaction), capacidades de particionamiento para escalabilidad horizontal, y soporte técnico empresarial.

Desafíos: Costo de licenciamiento significativo, curva de aprendizaje más pronunciada, particularidades como la ausencia de tipo BOOLEAN nativo (resuelta mediante conversión a NUMBER(1)), y complejidad en la configuración de alta disponibilidad.

Comparación con PostgreSQL: PostgreSQL ofrece características similares sin costo de licencia, soporte nativo de tipos de datos más amplio (incluyendo BOOLEAN, JSON, arrays), y una comunidad activa. Sin embargo, Oracle proporciona mejor rendimiento en operaciones de escritura intensiva y transacciones complejas, según benchmarks de TPC-C.

Comparación con SQL Server: SQL Server presenta mejor integración con ecosistemas Microsoft y herramientas de desarrollo .NET. La elección de Oracle se justifica por su independencia de plataforma y mayor presencia en el sector público colombiano.

7.3. Entity Framework Core: Ventajas y Limitaciones

El uso de Entity Framework Core como ORM proporciona abstracción de base de datos y productividad en desarrollo, pero introduce consideraciones importantes:

Ventajas: Code-first migrations facilitan el versionamiento del esquema, LINQ proporciona type safety en consultas, y el patrón Unit of Work simplifica el manejo transaccional. La portabilidad multi-proveedor resultó exitosa en pruebas con cuatro DBMS diferentes.

Limitaciones: El overhead de abstracción genera consultas SQL menos optimizadas que las escritas manualmente. En consultas críticas de rendimiento, puede ser necesario usar SQL directo mediante FromSqlRaw. El seguimiento de cambios (change tracking) consume memoria significativa en operaciones masivas, mitigado mediante AsNoTracking.

7.4. Escalabilidad y Consideraciones Futuras

El diseño actual soporta adecuadamente la carga esperada inicial (estimada en 1,000 citas diarias), pero requiere consideraciones para escalabilidad:

Particionamiento horizontal: Las tablas de citas y auditoría son candidatas para particionamiento por rango de fechas, separando datos históricos de activos.

Réplicas de lectura: Implementar réplicas de solo lectura para consultas de reportes y análisis, descargando la base de datos principal.

Caché distribuido: Datos de catálogo (departamentos, ciudades, servicios) pueden almacenarse en Redis para reducir consultas a la base de datos.

7.5. Seguridad y Cumplimiento Normativo

El modelo RBAC implementado proporciona control de acceso granular, cumpliendo con principios de least privilege. Sin embargo, para cumplir completamente con la Ley 1581 de 2012 (Protección de Datos Personales en Colombia), se requieren implementaciones adicionales:

Cifrado de campos sensibles (documentos de identidad, información de contacto).

Mecanismos de anonimización para cumplir con el derecho al olvido.

Logs de acceso a datos personales para auditorías de cumplimiento.

8. Conclusiones

La arquitectura de base de datos implementada para el sistema de agendamiento de citas ElectroHuila demuestra la aplicabilidad exitosa de principios de diseño relacional, patrones arquitectónicos modernos, y tecnologías ORM en sistemas empresariales del sector público. Las conclusiones principales de este trabajo son:

8.1. Logros Técnicos

El esquema relacional de 22 tablas normalizadas a 3NF proporciona una base sólida para la integridad de datos y mantenibilidad del sistema. La combinación de normalización rigurosa con desnormalizaciones estratégicas logra un balance efectivo entre consistencia de datos y rendimiento de consultas.

La implementación mediante Entity Framework Core 9.0 demostró capacidad de abstracción efectiva, permitiendo portabilidad entre cuatro gestores de bases de datos diferentes (Oracle, SQL Server, PostgreSQL, MySQL) con un único modelo de dominio. Esto reduce significativamente el riesgo tecnológico y facilita migraciones futuras.

Las características de auditoría temporal y soft deletes implementadas cumplen con requerimientos de trazabilidad del sector público, proporcionando capacidad de análisis retrospectivo sin comprometer el rendimiento de operaciones transaccionales.

8.2. Aplicabilidad del Modelo RBAC

El modelo de control de acceso basado en roles (RBAC) implementado mediante las tablas USERS, ROLES, PERMISSIONS, USER_ROLES y ROLE_PERMISSIONS proporciona flexibilidad y granularidad adecuadas para gestionar permisos en un sistema multiusuario complejo. Este diseño es extensible para incorporar permisos adicionales sin modificar el esquema base.

8.3. Optimización y Rendimiento

Las pruebas de rendimiento confirmaron que el uso estratégico de índices, consultas AsNoTracking, y patrones de carga eager vs. lazy produce tiempos de respuesta aceptables para el volumen de transacciones esperado. El overhead introducido por Entity Framework Core es compensado por la productividad ganada en desarrollo y mantenimiento.

8.4. Consideraciones para Oracle Database

Las particularidades de Oracle Database, especialmente la conversión de tipos booleanos a NUMBER(1) y las convenciones de nomenclatura en mayúsculas, requieren configuraciones específicas que fueron implementadas exitosamente mediante Fluent API. Estas adaptaciones no comprometieron la portabilidad del modelo de dominio.

8.5. Recomendaciones para Trabajo Futuro

Futuras iteraciones del sistema deberían considerar:

- Implementación de particionamiento horizontal en tablas de alto crecimiento (APPOINTMENTS, AUDIT_LOGS).
- Incorporación de cifrado a nivel de columna para datos sensibles, cumpliendo con la Ley 1581 de 2012.
- Evaluación de caché distribuido (Redis) para datos de catálogo de alta frecuencia de lectura.
- Implementación de estrategias de respaldo y recuperación ante desastres específicas para Oracle Database.
- Análisis de migración a microservicios con bases de datos descentralizadas por contexto acotado.

8.6. Aporte al Conocimiento

Este trabajo aporta un caso de estudio documentado de diseño de base de datos para sistemas de agendamiento PQR en el sector público colombiano, implementado con tecnologías actuales (.NET,

Entity Framework Core, Oracle Database). El enfoque de Arquitectura Limpia aplicado a la capa de persistencia es replicable en dominios similares.

La experiencia adquirida en la adaptación de Entity Framework Core para Oracle Database, particularmente en el manejo de tipos booleanos y convenciones de nomenclatura, constituye conocimiento práctico valioso para la comunidad de desarrolladores .NET que trabajan con este gestor de bases de datos.