| | |
|---|---|
| ***Sesión*** | ***22*** |
| ***Tema*** | Definición de tipos de datos, tablas y |
| ***Propósito*** | Dar a conocer a los participantes los fundamentos, técnicas y procedimientos de definición de tipo de datos, tablas y restricciones al implementar bases de datos, mediante la correspondiente presentación y demostración del docente en una sesión expositiva-demostrativa. |
| ***Fecha*** | C.20.11.2025 |
| ***Hora*** | 18:30 |

# 1. Tipos de Datos de Usuario (UDT)

## Tipos de Datos Básicos Personalizados

```
-- Crear tipos de datos básicos
CREATE DOMAIN email_type AS VARCHAR(255)
CHECK (VALUE ~ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');

CREATE DOMAIN phone_type AS VARCHAR(15)
CHECK (VALUE ~ '^\+?[0-9]{10,15}$');

CREATE DOMAIN percentage_type AS DECIMAL(5,2)
CHECK (VALUE BETWEEN 0 AND 100);

CREATE DOMAIN positive_int AS INTEGER
CHECK (VALUE > 0);
```

## Tipos de Datos Compuestos (PostgreSQL)

```
-- Crear tipo de dato compuesto para dirección
CREATE TYPE address_type AS (
    calle VARCHAR(100),
    ciudad VARCHAR(50),
    estado VARCHAR(50),
    codigo_postal VARCHAR(10),
```

```
    pais VARCHAR(50)
);

-- Crear tipo de dato compuesto para información de contacto
CREATE TYPE contact_info_type AS (
    telefono phone_type,
    email email_type,
    direccion address_type
);
```

# 2. Definición de Tablas con Tipos Personalizados

## Tabla de Clientes

```
CREATE TABLE clientes (
    cliente_id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    contacto contact_info_type NOT NULL,
    fecha_registro DATE DEFAULT CURRENT_DATE,
    activo BOOLEAN DEFAULT TRUE
);
```

## Tabla de Productos

```
CREATE TABLE productos (
    producto_id SERIAL PRIMARY KEY,
    nombre VARCHAR(200) NOT NULL,
    descripcion TEXT,
    precio DECIMAL(10,2) NOT NULL CHECK (precio > 0),
    porcentaje_descuento percentage_type DEFAULT 0,
    stock positive_int DEFAULT 0,
    categoria_id INTEGER
```

```
);
```

## Tabla de Pedidos

```
CREATE TABLE pedidos (
    pedido_id SERIAL PRIMARY KEY,
    cliente_id INTEGER NOT NULL,
    fecha_pedido TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    estado_pedido VARCHAR(20)
        CHECK (estado_pedido IN ('pendiente', 'procesando', 'enviado', 'entregado',
'cancelado')),
    total DECIMAL(12,2) NOT NULL CHECK (total >= 0),

    -- Restricción de clave foránea
    CONSTRAINT fk_cliente
        FOREIGN KEY (cliente_id)
        REFERENCES clientes(cliente_id)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);
```

# 3. Restricciones Avanzadas

## Restricciones a Nivel de Tabla

```
-- Tabla con múltiples restricciones
CREATE TABLE empleados (
    empleado_id SERIAL PRIMARY KEY,
    dni VARCHAR(20) UNIQUE NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    email email_type UNIQUE NOT NULL,
    telefono phone_type,
    salario DECIMAL(10,2) NOT NULL CHECK (salario >= 0),
```

```
    departamento_id INTEGER NOT NULL,
    jefe_id INTEGER,
    fecha_contratacion DATE NOT NULL,
    fecha_nacimiento DATE NOT NULL,

    -- Restricciones de clave foránea
    CONSTRAINT fk_departamento
        FOREIGN KEY (departamento_id)
        REFERENCES departamentos(departamento_id),

    CONSTRAINT fk_jefe
        FOREIGN KEY (jefe_id)
        REFERENCES empleados(empleado_id),

    -- Restricción CHECK compleja
    CONSTRAINT chk_fechas
        CHECK (fecha_contratacion > fecha_nacimiento),

    -- Restricción CHECK con función
    CONSTRAINT chk_edad_contratacion
        CHECK (EXTRACT(YEAR FROM AGE(fecha_contratacion, fecha_nacimiento)) >= 18)
);
```

## Restricciones de Unicidad Compuesta

```
CREATE TABLE cursos (
    curso_id SERIAL PRIMARY KEY,
    codigo_curso VARCHAR(20) NOT NULL,
    nombre_curso VARCHAR(200) NOT NULL,
    semestre VARCHAR(10) NOT NULL,
    año INTEGER NOT NULL CHECK (año >= 2000),

    -- Restricción de unicidad compuesta
    CONSTRAINT uk_curso_unico
        UNIQUE (codigo_curso, semestre, año)
);
```

# 4. Tablas con Enumeraciones

```
-- Crear tipo enumerado
CREATE TYPE estado_pedido_type AS ENUM (
    'pendiente',
    'confirmado',
    'en_proceso',
    'enviado',
    'entregado',
    'cancelado'
);

CREATE TYPE prioridad_type AS ENUM ('baja', 'media', 'alta', 'urgente');

-- Usar el tipo enumerado en una tabla
CREATE TABLE ordenes_trabajo (
    orden_id SERIAL PRIMARY KEY,
    descripcion TEXT NOT NULL,
    estado estado_pedido_type DEFAULT 'pendiente',
    prioridad prioridad_type DEFAULT 'media',
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    fecha_limite DATE,

    CONSTRAINT chk_fechas_orden
        CHECK (fecha_limite >= CURRENT_DATE)
);
```

# 5. Ejemplos de Inserción y Consulta

## Insertar Datos

```
-- Insertar en tabla con tipos compuestos (PostgreSQL)
INSERT INTO clientes (nombre, apellido, contacto)
VALUES (
    'María',
    'González',
```

```
        ROW('+34123456789', 'maria.gonzalez@email.com',
            ROW('Calle Principal 123', 'Madrid', 'Madrid', '28001', 'España')
        )::contact_info_type
);

-- Insertar en tabla con tipos básicos personalizados
INSERT INTO productos (nombre, precio, porcentaje_descuento, stock)
VALUES ('Laptop Gaming', 1200.00, 15.50, 25);
```

## Consultar Datos

```
-- Consultar campos de tipos compuestos
SELECT
    nombre,
    apellido,
    (contacto).email as email,
    ((contacto).direccion).ciudad as ciudad
FROM clientes;

-- Consultar con tipos enumerados
SELECT *
FROM ordenes_trabajo
WHERE estado = 'pendiente'
    OR prioridad = 'alta';
```

# 6. Ventajas de Usar Tipos de Datos Personalizados

- **Consistencia**: Garantizan formato uniforme en toda la base de datos
- **Mantenibilidad**: Cambios se aplican globalmente
- **Validación**: Restricciones integradas en el tipo de dato
- **Legibilidad**: Código más claro y comprensible
- **Reutilización**: Pueden usarse en múltiples tablas

# 1. Definición de Tipos de Datos Personalizados para el Hotel

```
-- Tipos de datos básicos personalizados
CREATE DOMAIN hotel_email AS VARCHAR(255)
CHECK (VALUE ~ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$');

CREATE DOMAIN phone_number AS VARCHAR(20)
CHECK (VALUE ~ '^\+?[0-9]{10,15}$');

CREATE DOMAIN percentage_value AS DECIMAL(5,2)
CHECK (VALUE BETWEEN 0 AND 100);

CREATE DOMAIN positive_money AS DECIMAL(10,2)
CHECK (VALUE >= 0);

CREATE DOMAIN room_number_type AS VARCHAR(10)
CHECK (VALUE ~ '^[0-9]{1,3}[A-Z]?$');

CREATE DOMAIN passport_number AS VARCHAR(20)
CHECK (VALUE ~ '^[A-Z0-9]{6,20}$');

-- Tipos enumerados
CREATE TYPE reservation_status_type AS ENUM (
    'pendiente',
    'confirmada',
    'check-in',
    'en_progreso',
    'check-out',
    'cancelada',
    'no_show'
);
```

```
CREATE TYPE room_status_type AS ENUM (
    'disponible',
    'ocupada',
    'mantenimiento',
    'limpieza',
    'reservada'
);

CREATE TYPE bed_type_enum AS ENUM (
    'individual',
    'doble',
    'queen',
    'king',
    'cama_extra'
);

CREATE TYPE payment_status_type AS ENUM (
    'pendiente',
    'parcial',
    'completado',
    'reembolsado',
    'fallido'
);

-- Tipos compuestos
CREATE TYPE address_type AS (
    calle VARCHAR(100),
    ciudad VARCHAR(50),
    estado_provincia VARCHAR(50),
    codigo_postal VARCHAR(15),
    pais VARCHAR(50)
);
```

```
CREATE TYPE guest_document_type AS (
    tipo_documento VARCHAR(20),
    numero_documento passport_number,
    pais_emision VARCHAR(50),
    fecha_vencimiento DATE
);

CREATE TYPE payment_method_type AS (
    tipo_pago VARCHAR(20),
    ultimos_digitos VARCHAR(4),
    nombre_titular VARCHAR(100)
);
```

# 2. Estructura de Tablas del Sistema Hotelero

## Tabla de Huéspedes

```
CREATE TABLE huespedes (
    huesped_id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    email hotel_email UNIQUE NOT NULL,
    telefono phone_number NOT NULL,
    fecha_nacimiento DATE NOT NULL,
    nacionalidad VARCHAR(50) NOT NULL,
    documento guest_document_type NOT NULL,
    direccion address_type,
    preferencias JSONB, -- Para almacenar preferencias como smoking, piso, etc.
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    nivel_miembro VARCHAR(20) DEFAULT 'estandar',

    CONSTRAINT chk_edad_minima
        CHECK (EXTRACT(YEAR FROM AGE(fecha_nacimiento)) >= 18)
);
```

# Tabla de Tipos de Habitación

```
CREATE TABLE tipos_habitacion (
    tipo_habitacion_id SERIAL PRIMARY KEY,
    nombre VARCHAR(50) UNIQUE NOT NULL,
    descripcion TEXT,
    capacidad_adultos INTEGER NOT NULL CHECK (capacidad_adultos > 0),
    capacidad_ninos INTEGER DEFAULT 0 CHECK (capacidad_ninos >= 0),
    tamano_m2 DECIMAL(6,2) CHECK (tamano_m2 > 0),
    camas JSONB NOT NULL, -- Ej: [{"tipo": "king", "cantidad": 1}, {"tipo": "individual", "cantidad": 1}]
    comodidades TEXT[], -- Array de comodidades: {'wifi', 'tv', 'minibar', 'vista_mar'}
    tarifa_base positive_money NOT NULL,
    tarifa_fin_semana positive_money,
    politicas_cancelacion TEXT
);
```

# Tabla de Habitaciones

```
CREATE TABLE habitaciones (
    habitacion_id SERIAL PRIMARY KEY,
    numero_habitacion room_number_type UNIQUE NOT NULL,
    tipo_habitacion_id INTEGER NOT NULL,
    piso INTEGER NOT NULL CHECK (piso BETWEEN 1 AND 50),
    estado room_status_type DEFAULT 'disponible',
    ultima_limpieza TIMESTAMP,
    proximo_mantenimiento DATE,
    caracteristicas_especiales TEXT[], -- {'vista_mar', 'terraza', 'jacuzzi'}

    CONSTRAINT fk_tipo_habitacion
        FOREIGN KEY (tipo_habitacion_id)
        REFERENCES tipos_habitacion(tipo_habitacion_id)
        ON DELETE RESTRICT
);
```

# Tabla de Reservas

```
CREATE TABLE reservas (
    reserva_id SERIAL PRIMARY KEY,
    numero_reserva VARCHAR(20) UNIQUE NOT NULL,
    huesped_id INTEGER NOT NULL,
    fecha_entrada DATE NOT NULL,
    fecha_salida DATE NOT NULL,
    noches INTEGER GENERATED ALWAYS AS (fecha_salida - fecha_entrada) STORED,
    numero_adultos INTEGER NOT NULL CHECK (numero_adultos > 0),
    numero_ninos INTEGER DEFAULT 0 CHECK (numero_ninos >= 0),
    estado reservation_status_type DEFAULT 'pendiente',
    tipo_tarifa VARCHAR(20) DEFAULT 'estandar', -- 'estandar', 'promocional', 'corporativa'
    total_reserva positive_money,
    senia positive_money DEFAULT 0,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    notas TEXT,

    CONSTRAINT fk_huesped
        FOREIGN KEY (huesped_id)
        REFERENCES huespedes(huesped_id),

    CONSTRAINT chk_fechas_validas
        CHECK (fecha_salida > fecha_entrada),

    CONSTRAINT chk_max_capacidad
        CHECK (numero_adultos + numero_ninos <= 6) -- Capacidad máxima por reserva
);
```

# Tabla de Detalles de Reserva (Habitaciones asignadas)

```
CREATE TABLE detalles_reserva (
    detalle_id SERIAL PRIMARY KEY,
    reserva_id INTEGER NOT NULL,
    habitacion_id INTEGER NOT NULL,
    tarifa_aplicada positive_money NOT NULL,
    comisiones percentage_value DEFAULT 0,
```

```
    servicios_extra JSONB, -- {'desayuno': true, 'parking': false, 'spa': true}
    total_detalle positive_money GENERATED ALWAYS AS (
        tarifa_aplicada * (SELECT noches FROM reservas WHERE reserva_id =
detalles_reserva.reserva_id)
    ) STORED,

    CONSTRAINT fk_reserva
        FOREIGN KEY (reserva_id)
        REFERENCES reservas(reserva_id)
        ON DELETE CASCADE,

    CONSTRAINT fk_habitacion
        FOREIGN KEY (habitacion_id)
        REFERENCES habitaciones(habitacion_id),

    CONSTRAINT uk_reserva_habitacion
        UNIQUE (reserva_id, habitacion_id)
);
```

## Tabla de Pagos

```
CREATE TABLE pagos (
    pago_id SERIAL PRIMARY KEY,
    reserva_id INTEGER NOT NULL,
    monto positive_money NOT NULL,
    metodo_pago payment_method_type NOT NULL,
    estado_pago payment_status_type DEFAULT 'pendiente',
    fecha_pago TIMESTAMP,
    referencia_pago VARCHAR(50),
    descripcion TEXT,

    CONSTRAINT fk_reserva
        FOREIGN KEY (reserva_id)
        REFERENCES reservas(reserva_id)
);
```

## Tabla de Servicios Adicionales

```
CREATE TABLE servicios (
    servicio_id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    descripcion TEXT,
    precio positive_money NOT NULL,
    categoria VARCHAR(50) NOT NULL, -- 'restaurante', 'spa', 'actividades', 'otros'
    disponible BOOLEAN DEFAULT TRUE
);

CREATE TABLE servicios_reserva (
    servicio_reserva_id SERIAL PRIMARY KEY,
    reserva_id INTEGER NOT NULL,
    servicio_id INTEGER NOT NULL,
    fecha_servicio DATE NOT NULL,
    cantidad INTEGER DEFAULT 1 CHECK (cantidad > 0),
    precio_unitario positive_money NOT NULL,
    total_servicio positive_money GENERATED ALWAYS AS (cantidad * precio_unitario) STORED,

    CONSTRAINT fk_reserva
        FOREIGN KEY (reserva_id)
        REFERENCES reservas(reserva_id),

    CONSTRAINT fk_servicio
        FOREIGN KEY (servicio_id)
        REFERENCES servicios(servicio_id)
);
```

# 3. Inserciones de Ejemplo

```
-- Insertar tipos de habitación
INSERT INTO tipos_habitacion (nombre, descripcion, capacidad_adultos, capacidad_ninos,
tamano_m2, camas, comodidades, tarifa_base, tarifa_fin_semana)
VALUES
('Estándar', 'Habitación estándar con cama doble', 2, 1, 25.0,
  '[{"tipo": "doble", "cantidad": 1}]',
```

```
  '{"wifi", "tv", "ac"}', 100.00, 120.00),

('Suite', 'Suite con sala separada y vista al mar', 3, 2, 45.0,
  '[{"tipo": "king", "cantidad": 1}, {"tipo": "individual", "cantidad": 1}]',
  '{"wifi", "tv", "ac", "minibar", "vista_mar"}', 200.00, 250.00);

-- Insertar habitaciones
INSERT INTO habitaciones (numero_habitacion, tipo_habitacion_id, piso,
caracteristicas_especiales)
VALUES
('101', 1, 1, '{}'),
('102', 1, 1, '{}'),
('201', 2, 2, '{"vista_mar", "terraza"}');

-- Insertar huésped
INSERT INTO huespedes (nombre, apellido, email, telefono, fecha_nacimiento, nacionalidad,
documento)
VALUES (
        'Carlos',
        'Martínez',
        'carlos.martinez@email.com',
        '+34123456789',
        '1985-05-15',
        'Española',
        ROW('pasaporte', 'AB123456', 'España', '2030-12-31')::guest_document_type
);

-- Crear reserva
INSERT INTO reservas (numero_reserva, huesped_id, fecha_entrada, fecha_salida,
numero_adultos, numero_ninos, total_reserva)
VALUES ('RES2024001', 1, '2024-01-15', '2024-01-20', 2, 1, 600.00);

-- Asignar habitación a reserva
INSERT INTO detalles_reserva (reserva_id, habitacion_id, tarifa_aplicada, servicios_extra)
VALUES (1, 1, 120.00, '{"desayuno": true, "parking": true}');
```

# 4. Consultas Útiles para la Gestión Hotelera

## Reservas Activas

```
SELECT
    r.numero_reserva,
    h.nombre || ' ' || h.apellido as huesped,
    r.fecha_entrada,
    r.fecha_salida,
    r.estado,
    ha.numero_habitacion,
    r.total_reserva
FROM reservas r
JOIN huespedes h ON r.huesped_id = h.huesped_id
JOIN detalles_reserva dr ON r.reserva_id = dr.reserva_id
JOIN habitaciones ha ON dr.habitacion_id = ha.habitacion_id
WHERE r.estado IN ('confirmada', 'check-in', 'en_progreso')
ORDER BY r.fecha_entrada;
```

## Ocupación por Día

```
SELECT
    fecha,
    COUNT(*) as habitaciones_ocupadas,
    (SELECT COUNT(*) FROM habitaciones) as total_habitaciones
FROM (
    SELECT GENERATE_SERIES(
        fecha_entrada,
        fecha_salida - INTERVAL '1 day',
        '1 day'::interval
    )::date as fecha
    FROM reservas
    WHERE estado IN ('confirmada', 'check-in', 'en_progreso')
) fechas_ocupacion
GROUP BY fecha
ORDER BY fecha;
```

## Ingresos por Tipo de Habitación

```
SELECT
    th.nombre as tipo_habitacion,
    COUNT(dr.detalle_id) as numero_reservas,
    SUM(dr.total_detalle) as ingresos_totales,
    AVG(dr.tarifa_aplicada) as tarifa_promedio
FROM detalles_reserva dr
JOIN habitaciones h ON dr.habitacion_id = h.habitacion_id
JOIN tipos_habitacion th ON h.tipo_habitacion_id = th.tipo_habitacion_id
JOIN reservas r ON dr.reserva_id = r.reserva_id
WHERE r.fecha_entrada >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY th.tipo_habitacion_id, th.nombre
ORDER BY ingresos_totales DESC;
```

# 5. Ventajas de esta Implementación

- **Tipos de datos seguros**: Validación automática de emails, teléfonos, etc.
- **Estructura flexible**: JSONB para preferencias y servicios extra
- **Consistencia**: Enumerados para estados predefinidos
- **Rendimiento**: Tipos compuestos para datos relacionados
- **Escalabilidad**: Fácil añadir nuevos tipos de habitación o servicios

# 6. Procedimientos Almacenados CRUD

## CRUD para Huéspedes

```
-- Crear huésped
CREATE OR REPLACE PROCEDURE crear_huesped(
    p_nombre VARCHAR(100),
    p_apellido VARCHAR(100),
    p_email VARCHAR(255),
    p_telefono VARCHAR(20),
    p_fecha_nacimiento DATE,
    p_nacionalidad VARCHAR(50),
    p_tipo_documento VARCHAR(20),
    p_numero_documento VARCHAR(20),
    p_pais_emision VARCHAR(50),
    p_fecha_vencimiento DATE,
    INOUT p_huesped_id INTEGER DEFAULT NULL
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO huespedes (
        nombre, apellido, email, telefono, fecha_nacimiento,
        nacionalidad, documento
    ) VALUES (
        p_nombre, p_apellido, p_email, p_telefono, p_fecha_nacimiento,
        p_nacionalidad,
        ROW(p_tipo_documento, p_numero_documento, p_pais_emision,
p_fecha_vencimiento)::guest_document_type
    )
    RETURNING huesped_id INTO p_huesped_id;
    RAISE NOTICE 'Huésped creado con ID: %', p_huesped_id;
END;
$$;
```

```
-- Leer huésped
CREATE OR REPLACE FUNCTION leer_huesped(p_huesped_id INTEGER)
RETURNS TABLE(
    id INTEGER,
    nombre_completo TEXT,
    email VARCHAR,
    telefono VARCHAR,
    nacionalidad VARCHAR,
    tipo_documento VARCHAR,
    numero_documento VARCHAR
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        h.huesped_id,
        h.nombre || ' ' || h.apellido,
        h.email,
        h.telefono,
        h.nacionalidad,
        (h.documento).tipo_documento,
        (h.documento).numero_documento
    FROM huespedes h
    WHERE h.huesped_id = p_huesped_id;
END;
$$;

-- Actualizar huésped
CREATE OR REPLACE PROCEDURE actualizar_huesped(
    p_huesped_id INTEGER,
    p_telefono VARCHAR(20) DEFAULT NULL,
    p_email VARCHAR(255) DEFAULT NULL,
    p_preferencias JSONB DEFAULT NULL
)
LANGUAGE plpgsql
AS $$
```

```
BEGIN
    UPDATE huespedes
    SET
        telefono = COALESCE(p_telefono, telefono),
        email = COALESCE(p_email, email),
        preferencias = COALESCE(p_preferencias, preferencias)
    WHERE huesped_id = p_huesped_id;

    IF FOUND THEN
        RAISE NOTICE 'Huésped % actualizado correctamente', p_huesped_id;
    ELSE
        RAISE EXCEPTION 'Huésped con ID % no encontrado', p_huesped_id;
    END IF;
END;
$$;

-- Eliminar huésped (soft delete)
CREATE OR REPLACE PROCEDURE eliminar_huesped(p_huesped_id INTEGER)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Verificar si el huésped tiene reservas activas
    IF EXISTS (
        SELECT 1 FROM reservas
        WHERE huesped_id = p_huesped_id
        AND estado IN ('pendiente', 'confirmada', 'check-in')
    ) THEN
        RAISE EXCEPTION 'No se puede eliminar huésped con reservas activas';
    END IF;
    -- Marcar como inactivo en lugar de eliminar
    UPDATE huespedes
    SET email = 'inactive_' || huesped_id || '_' || email,
        telefono = NULL
    WHERE huesped_id = p_huesped_id;

    RAISE NOTICE 'Huésped % marcado como inactivo', p_huesped_id;
END;
$$;
```

# CRUD para Reservas

```
-- Crear reserva
CREATE OR REPLACE PROCEDURE crear_reserva(
    p_huesped_id INTEGER,
    p_fecha_entrada DATE,
    p_fecha_salida DATE,
    p_numero_adultos INTEGER,
    p_numero_ninos INTEGER DEFAULT 0,
    p_tipo_tarifa VARCHAR(20) DEFAULT 'estandar',
    p_notas TEXT DEFAULT NULL,
    INOUT p_reserva_id INTEGER DEFAULT NULL
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_noches INTEGER;
    v_numero_reserva VARCHAR(20);
    v_total_estimado DECIMAL(10,2);
BEGIN
    -- Calcular número de noches
    v_noches := p_fecha_salida - p_fecha_entrada;

    IF v_noches <= 0 THEN
        RAISE EXCEPTION 'La fecha de salida debe ser posterior a la fecha de entrada';
    END IF;

    -- Generar número de reserva único
    v_numero_reserva := 'RES' || TO_CHAR(CURRENT_DATE, 'YYYYMMDD') ||
                        LPAD((COALESCE((SELECT MAX(reserva_id) FROM reservas), 0) +
1)::TEXT, 4, '0');

    -- Calcular total estimado (se actualizará cuando se asignen habitaciones)
    v_total_estimado := 0;

    INSERT INTO reservas (
        numero_reserva, huesped_id, fecha_entrada, fecha_salida,
```

```
        numero_adultos, numero_ninos, tipo_tarifa, total_reserva, notas
    ) VALUES (
        v_numero_reserva, p_huesped_id, p_fecha_entrada, p_fecha_salida,
        p_numero_adultos, p_numero_ninos, p_tipo_tarifa, v_total_estimado, p_notas
    )
    RETURNING reserva_id INTO p_reserva_id;

    RAISE NOTICE 'Reserva % creada con ID: %', v_numero_reserva, p_reserva_id;
END;
$$;

-- Asignar habitación a reserva
CREATE OR REPLACE PROCEDURE asignar_habitacion_reserva(
    p_reserva_id INTEGER,
    p_habitacion_id INTEGER,
    p_tarifa_aplicada DECIMAL(10,2) DEFAULT NULL
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_tarifa_base DECIMAL(10,2);
    v_noches INTEGER;
    v_tarifa_final DECIMAL(10,2);
BEGIN
    -- Obtener tarifa base si no se proporciona
    IF p_tarifa_aplicada IS NULL THEN
        SELECT th.tarifa_base INTO v_tarifa_base
        FROM habitaciones h
        JOIN tipos_habitacion th ON h.tipo_habitacion_id = th.tipo_habitacion_id
        WHERE h.habitacion_id = p_habitacion_id;

        v_tarifa_final := v_tarifa_base;
    ELSE
        v_tarifa_final := p_tarifa_aplicada;
    END IF;

    -- Obtener número de noches
    SELECT noches INTO v_noches FROM reservas WHERE reserva_id = p_reserva_id;
```

```
    -- Insertar detalle de reserva
    INSERT INTO detalles_reserva (reserva_id, habitacion_id, tarifa_aplicada)
    VALUES (p_reserva_id, p_habitacion_id, v_tarifa_final);

    -- Actualizar estado de la habitación
    UPDATE habitaciones SET estado = 'reservada'
    WHERE habitacion_id = p_habitacion_id;

    -- Actualizar total de la reserva
    UPDATE reservas
    SET total_reserva = (
        SELECT SUM(tarifa_aplicada * v_noches)
        FROM detalles_reserva
        WHERE reserva_id = p_reserva_id
    )
    WHERE reserva_id = p_reserva_id;

    RAISE NOTICE 'Habitación % asignada a reserva %', p_habitacion_id, p_reserva_id;
END;
$$;

-- Check-in
CREATE OR REPLACE PROCEDURE realizar_checkin(
    p_reserva_id INTEGER
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Actualizar estado de la reserva
    UPDATE reservas
    SET estado = 'check-in'
    WHERE reserva_id = p_reserva_id;

    -- Actualizar estado de las habitaciones
    UPDATE habitaciones
    SET estado = 'ocupada'
    WHERE habitacion_id IN (
```

```
            SELECT habitacion_id
            FROM detalles_reserva
            WHERE reserva_id = p_reserva_id
        );

        -- Registrar fecha de check-in
        INSERT INTO auditoria_reservas (reserva_id, accion, detalles)
        VALUES (p_reserva_id, 'check-in', 'Check-in realizado ' || CURRENT_TIMESTAMP);

        RAISE NOTICE 'Check-in realizado para reserva %', p_reserva_id;
END;
$$;

-- Check-out
CREATE OR REPLACE PROCEDURE realizar_checkout(
        p_reserva_id INTEGER
)
LANGUAGE plpgsql
AS $$
DECLARE
        v_total_servicios DECIMAL(10,2);
BEGIN
        -- Calcular total de servicios adicionales
        SELECT COALESCE(SUM(total_servicio), 0) INTO v_total_servicios
        FROM servicios_reserva
        WHERE reserva_id = p_reserva_id;

        -- Actualizar estado de la reserva
        UPDATE reservas
        SET estado = 'check-out',
            total_reserva = total_reserva + v_total_servicios
        WHERE reserva_id = p_reserva_id;

        -- Liberar habitaciones
        UPDATE habitaciones
        SET estado = 'limpieza'
        WHERE habitacion_id IN (
            SELECT habitacion_id
```

```
            FROM detalles_reserva
            WHERE reserva_id = p_reserva_id
        );

        -- Registrar check-out
        INSERT INTO auditoria_reservas (reserva_id, accion, detalles)
        VALUES (p_reserva_id, 'check-out',
                    'Check-out realizado. Servicios adicionales: ' || v_total_servicios);

        RAISE NOTICE 'Check-out realizado para reserva %. Total servicios: %',
                p_reserva_id, v_total_servicios;
END;
$$;
```

# 2. Triggers y Funciones de Trigger

## Triggers para Auditoría

```
-- Tabla de auditoría para reservas
CREATE TABLE auditoria_reservas (
        auditoria_id SERIAL PRIMARY KEY,
        reserva_id INTEGER NOT NULL,
        accion VARCHAR(50) NOT NULL,
        fecha_auditoria TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        usuario VARCHAR(100) DEFAULT CURRENT_USER,
        detalles TEXT
);

-- Función para auditoría de reservas
CREATE OR REPLACE FUNCTION fn_auditoria_reservas()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
        IF TG_OP = 'INSERT' THEN
                INSERT INTO auditoria_reservas (reserva_id, accion, detalles)
```

```
        VALUES (NEW.reserva_id, 'INSERT',
                'Nueva reserva creada: ' || NEW.numero_reserva);
    ELSIF TG_OP = 'UPDATE' THEN
        IF OLD.estado != NEW.estado THEN
            INSERT INTO auditoria_reservas (reserva_id, accion, detalles)
            VALUES (NEW.reserva_id, 'UPDATE',
                    'Estado cambiado de ' || OLD.estado || ' a ' || NEW.estado);
        END IF;

        IF OLD.total_reserva != NEW.total_reserva THEN
            INSERT INTO auditoria_reservas (reserva_id, accion, detalles)
            VALUES (NEW.reserva_id, 'UPDATE',
                    'Total actualizado de ' || OLD.total_reserva || ' a ' ||
NEW.total_reserva);
        END IF;
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO auditoria_reservas (reserva_id, accion, detalles)
        VALUES (OLD.reserva_id, 'DELETE',
                'Reserva eliminada: ' || OLD.numero_reserva);
    END IF;

    RETURN COALESCE(NEW, OLD);
END;
$$;

-- Trigger para auditoría de reservas
CREATE TRIGGER tr_auditoria_reservas
    AFTER INSERT OR UPDATE OR DELETE ON reservas
    FOR EACH ROW
    EXECUTE FUNCTION fn_auditoria_reservas();
```

## Triggers para Validación de Disponibilidad

```
-- Función para validar disponibilidad de habitación
CREATE OR REPLACE FUNCTION fn_validar_disponibilidad_habitacion()
RETURNS TRIGGER
LANGUAGE plpgsql
```

```
AS $$
DECLARE
    v_fecha_entrada DATE;
    v_fecha_salida DATE;
    v_habitacion_ocupada BOOLEAN;
BEGIN
    -- Obtener fechas de la reserva
    SELECT fecha_entrada, fecha_salida INTO v_fecha_entrada, v_fecha_salida
    FROM reservas WHERE reserva_id = NEW.reserva_id;

    -- Verificar si la habitación está ocupada en esas fechas
    SELECT EXISTS(
        SELECT 1
        FROM detalles_reserva dr
        JOIN reservas r ON dr.reserva_id = r.reserva_id
        WHERE dr.habitacion_id = NEW.habitacion_id
        AND r.reserva_id != NEW.reserva_id
        AND r.estado IN ('confirmada', 'check-in', 'en_progreso')
        AND (v_fecha_entrada, v_fecha_salida) OVERLAPS (r.fecha_entrada, r.fecha_salida)
    ) INTO v_habitacion_ocupada;

    IF v_habitacion_ocupada THEN
        RAISE EXCEPTION 'La habitación % no está disponible en las fechas seleccionadas',
            NEW.habitacion_id;
    END IF;

    RETURN NEW;
END;
$$;

-- Trigger para validar disponibilidad
CREATE TRIGGER tr_validar_disponibilidad
    BEFORE INSERT ON detalles_reserva
    FOR EACH ROW
    EXECUTE FUNCTION fn_validar_disponibilidad_habitacion();
```

# Trigger para Actualización Automática de Estado

```
-- Función para actualizar estado de habitaciones automáticamente
CREATE OR REPLACE FUNCTION fn_actualizar_estado_habitacion()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    -- Cuando una reserva cambia a check-out, poner habitación en limpieza
    IF NEW.estado = 'check-out' AND OLD.estado != 'check-out' THEN
        UPDATE habitaciones
        SET estado = 'limpieza',
            ultima_limpieza = CURRENT_TIMESTAMP
        WHERE habitacion_id IN (
            SELECT habitacion_id
            FROM detalles_reserva
            WHERE reserva_id = NEW.reserva_id
        );
    END IF;

    -- Cuando una reserva se cancela, liberar habitaciones
    IF NEW.estado = 'cancelada' AND OLD.estado != 'cancelada' THEN
        UPDATE habitaciones
        SET estado = 'disponible'
        WHERE habitacion_id IN (
            SELECT habitacion_id
            FROM detalles_reserva
            WHERE reserva_id = NEW.reserva_id
        );
    END IF;

    RETURN NEW;
END;
$$;

-- Trigger para actualizar estado de habitaciones
CREATE TRIGGER tr_actualizar_estado_habitacion
```

```
    AFTER UPDATE OF estado ON reservas
    FOR EACH ROW
    EXECUTE FUNCTION fn_actualizar_estado_habitacion();
```

## Trigger para Cálculo Automático de Totales

```
-- Función para calcular totales automáticamente
CREATE OR REPLACE FUNCTION fn_calcular_totales_reserva()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
DECLARE
    v_total_habitaciones DECIMAL(10,2);
    v_total_servicios DECIMAL(10,2);
    v_noches INTEGER;
BEGIN
    -- Calcular noches
    v_noches := NEW.fecha_salida - NEW.fecha_entrada;

    -- Calcular total de habitaciones
    SELECT COALESCE(SUM(tarifa_aplicada * v_noches), 0) INTO v_total_habitaciones
    FROM detalles_reserva
    WHERE reserva_id = NEW.reserva_id;

    -- Calcular total de servicios
    SELECT COALESCE(SUM(total_servicio), 0) INTO v_total_servicios
    FROM servicios_reserva
    WHERE reserva_id = NEW.reserva_id;

    -- Actualizar total de la reserva
    NEW.total_reserva := v_total_habitaciones + v_total_servicios;

    RETURN NEW;
END;
$$;
```

```
-- Trigger para calcular totales cuando cambian las fechas
CREATE TRIGGER tr_calcular_totales_reserva
    BEFORE UPDATE OF fecha_entrada, fecha_salida ON reservas
    FOR EACH ROW
    EXECUTE FUNCTION fn_calcular_totales_reserva();
```

# 3. Ejemplos de Uso

```
-- Ejemplo de uso de los procedimientos
DO $$
DECLARE
    v_huesped_id INTEGER;
    v_reserva_id INTEGER;
BEGIN
    -- Crear un huésped
    CALL crear_huesped(
        'Ana', 'López', 'ana.lopez@email.com', '+34123456789',
        '1990-08-20', 'Mexicana', 'pasaporte', 'MX123456',
        'México', '2030-12-31', v_huesped_id
    );

    -- Crear una reserva
    CALL crear_reserva(
        v_huesped_id, '2024-02-01', '2024-02-05',
        2, 1, 'estandar', 'Habitación con vista al mar', v_reserva_id
    );

    -- Asignar habitación
    CALL asignar_habitacion_reserva(v_reserva_id, 2);

    -- Realizar check-in
    CALL realizar_checkin(v_reserva_id);

    RAISE NOTICE 'Proceso completado. Huésped ID: %, Reserva ID: %',
        v_huesped_id, v_reserva_id;
END;
$$;
```

```
-- Consultar auditoría
SELECT * FROM auditoria_reservas ORDER BY fecha_auditoria DESC LIMIT 5;
```

## 4. Ventajas de esta Implementación

- **Consistencia**: Los triggers aseguran que las reglas de negocio se apliquen automáticamente
- **Auditoría completa**: Seguimiento de todos los cambios importantes
- **Validación en tiempo real**: Previene operaciones inválidas
- **Automatización**: Cálculos automáticos de totales y actualizaciones de estado
- **Mantenibilidad**: Lógica centralizada en procedimientos almacenados

---

# CASO PRÁCTICO:

## SISTEMA DE GESTIÓN PARA "HOTEL PARAÍSO TROPICAL"

### Enunciado del Caso

**Hotel Paraíso Tropical** es un hotel boutique de 4 estrellas que requiere modernizar su sistema de gestión. Actualmente manejan todo mediante planillas Excel y registros en papel, lo que genera problemas de sobre-reservas, pérdida de información y dificultades en la gestión de servicios.

# 1. Análisis de Requerimientos

## Requerimientos Funcionales:

### (a)Gestión de Huéspedes

- Registro completo de información personal y documentos
- Historial de estancias
- Preferencias y restricciones alimentarias

### (b)Gestión de Reservas

- Reservas online y telefónicas
- Múltiples tipos de habitación
- Temporadas alta/media/baja con tarifas diferenciadas
- Políticas de cancelación flexibles

### (c)Gestión de Habitaciones

- Control de disponibilidad en tiempo real
- Estados (disponible, ocupada, mantenimiento, limpieza)
- Características específicas por habitación

### (d)Servicios Adicionales

- Spa, restaurante, tours, transporte
- Cargo directo a la habitación
- Gestión de eventos y conferencias

### (e)Facturación y Pagos

- Múltiples métodos de pago
- Facturación por servicios
- Cálculo automático de impuestos (16% IVA)

### (f)Reportes y Analytics

- Ocupación hotelera
- Ingresos por servicios
- Clientes frecuentes
- Temporadas más rentables

**Requerimientos No Funcionales:**

- Base de datos PostgreSQL 12+
- Backup automático diario
- 100 usuarios concurrentes
- Respuesta en consultas < 3 segundos
- Auditoría completa de cambios

## 2. Diseño Conceptual (Modelo Entidad-Relación)

text

```
[HUÉSPED] ←→ [RESERVA] ←→ [HABITACIÓN]
      ↓                    ↓                ↓
[PAGO]      [SERVICIO]  [TIPO_HABITACIÓN]
      ↓                    ↓
[FACTURA]    [CATEGORÍA_SERVICIO]
```

## 3. Diseño Lógico

**Entidades Principales:**

- **HUÉSPEDES** (huesped_id, nombre, email, telefono, documento)
- **RESERVAS** (reserva_id, huesped_id, fecha_entrada, fecha_salida, estado)
- **HABITACIONES** (habitacion_id, tipo_id, numero, piso, estado)
- **TIPOS_HABITACION** (tipo_id, nombre, capacidad, tarifa_base)
- **SERVICIOS** (servicio_id, nombre, precio, categoria)
- **PAGOS** (pago_id, reserva_id, monto, metodo, estado)

# 4. Actividades de Implementación

## Fase 1: Definición de Esquema y Tipos de Datos

### **Tarea 1.1:** Crear tipos de datos personalizados

*-- Implementar tipos para:*
*-- • email_hotel (validación de email)*
*-- • telefono_movil (formato internacional)*
*-- • tipo_documento (enum: pasaporte, dni, licencia)*
*-- • estado_reserva (enum con todos los estados)*
*-- • tipo_habitacion_enum (estándar, suite, familiar, etc.)*

### **Tarea 1.2:** Crear tablas principales

*-- Implementar tablas para:*
*-- • huespedes*
*-- • reservas*
*-- • habitaciones*
*-- • tipos_habitacion*
*-- • servicios*
*-- • pagos*

## Fase 2: Restricciones e Integridad Referencial

### **Tarea 2.1:** Implementar constraints CHECK

*-- Validaciones para:*
*-- • Fecha salida > fecha entrada*
*-- • Capacidad máxima por habitación*
*-- • Precios positivos*
*-- • Edad mínima 18 años para reservar*

### **Tarea 2.2:** Claves foráneas y relaciones

*-- Establecer relaciones entre:*
*-- • Reservas → Huéspedes*
*-- • Reservas → Habitaciones*
*-- • Pagos → Reservas*

*-- • Servicios_Reserva → Reservas y Servicios*

## Fase 3: Procedimientos Almacenados

### **Tarea 3.1:** CRUD para entidades principales

*-- Crear procedimientos para:*
*-- • sp_crear_reserva_completa*
*-- • sp_realizar_checkin*
*-- • sp_realizar_checkout*
*-- • sp_agregar_servicio_habitacion*
*-- • sp_generar_factura*

### **Tarea 3.2:** Funciones de negocio

*-- Implementar funciones para:*
*-- • fn_calcular_total_reserva*
*-- • fn_verificar_disponibilidad*
*-- • fn_obtener_historial_huesped*
*-- • fn_calcular_ocupacion_periodo*

## Fase 4: Triggers y Auditoría

### **Tarea 4.1:** Triggers de validación

*-- Crear triggers para:*
*-- • tr_validar_fechas_reserva*
*-- • tr_evitar_doble_reserva*
*-- • tr_actualizar_estado_habitacion*
*-- • tr_calcular_totales_automaticos*

### **Tarea 4.2:** Sistema de auditoría

*-- Implementar:*
*-- • tabla_auditoria_reservas*
*-- • trigger_auditoria_cambios*
*-- • historial_estados_reserva*

### Fase 5: Datos de Prueba y Consultas

**Tarea 5.1:** Poblado de datos de prueba

*-- Insertar:*
*-- • 10 tipos de habitación*
*-- • 50 habitaciones*
*-- • 100 huéspedes de prueba*
*-- • 200 reservas de diferentes estados*
*-- • 500 servicios consumidos*

**Tarea 5.2:** Consultas de reporting

*-- Desarrollar consultas para:*
*-- • Ocupación por día/mes*
*-- • Ingresos por servicio*
*-- • Huéspedes frecuentes*
*-- • Habitaciones más rentables*
*-- • Temporadas alta/media/baja*

## 5. Criterios de Evaluación

### Completitud (40%)

- Todos los tipos de datos personalizados implementados
- Tablas principales creadas con constraints
- Relaciones establecidas correctamente
- Procedimientos CRUD funcionando
- Triggers de auditoría operativos

### Calidad del Código (30%)

- Nomenclatura consistente
- Comentarios explicativos
- Manejo de errores en procedimientos
- Optimización de consultas

**Funcionalidad (20%)**

- Datos de prueba insertados correctamente
- Consultas de reporting generan resultados esperados
- Validaciones previenen datos inconsistentes

**Documentación (10%)**

- Diagrama entidad-relación
- Diccionario de datos
- Manual de procedimientos almacenados

## 6. Entregables Esperados

1. **Script SQL completo** con toda la implementación
2. **Diagrama ER** del modelo de datos
3. **Diccionario de datos** con descripción de tablas y campos
4. **Colección de consultas** de business intelligence
5. **Documentación técnica** de procedimientos y triggers

## 7. Ejemplo de Escenario de Prueba

*-- Escenario: Reserva compleja con múltiples servicios*
*-- 1. Crear huésped internacional*
*-- 2. Hacer reserva para 5 noches, 2 adultos, 1 niño*
*-- 3. Asignar suite familiar*
*-- 4. Agregar servicios: desayuno diario, spa, tour*
*-- 5. Realizar check-in*
*-- 6. Agregar servicio de restaurante durante la estancia*
*-- 7. Realizar check-out*
*-- 8. Generar factura con desglose*
*-- 9. Procesar pago con tarjeta*

---

## Preguntas Guía para el Desarrollo:

(a) ¿Cómo manejarías las temporadas con tarifas diferenciadas?
(b) ¿Qué estrategia implementarías para evitar overbooking?
(c) ¿Cómo gestionarías cambios de habitación durante la estancia?
(d) ¿Qué información sería crucial para el dashboard de gestión?
(e) ¿Cómo implementarías un sistema de fidelización?

### Bibliografía Recomendada

• "Fundamentos de Bases de Datos" de Abraham Silberschatz, Henry F. Korth y S. Sudarshan

• "Sistemas de Bases de Datos: un enfoque práctico" de Thomas M. Connolly y Carolyn Begg

• "Desarrollo de Bases de Datos: casos prácticos desde el análisis a la implementación" de Dolores Cuadra, Elena Castro, Ana M. Iglesias

• "Tecnología y Diseño de Bases de Datos" de Marcos, C. Calero y B. Vela

• https://docs.oracle.com/en/database/