República Bolivariana de Venezuela
Ministerio del Poder Popular para la Educación Universitaria
Universidad Nacional Experimental Para Las Telecomunicaciones e Informática
PNF Ingeniería Informática
Base de Datos I
Sección 6B

FASE 3: DISEÑO LÓGICO Y FÍSICO DE LA BASE DE DATOS "CONSTRUCCIÓN RENTABLE"

Profesora:
Yuly Delgado

Ingeniero en Formación:
Mijael Engelmann
C.I:V-31.222.463

Introducción

La tercera fase del desarrollo de la base de datos para "Construcción Rentable" se enfoca en el diseño lógico y físico del sistema. Hasta este punto, hemos definido las tablas y relaciones clave (Fase 1) e implementado la inserción, consulta y eliminación de datos (Fase 2). Ahora, nos adentramos en la optimización del rendimiento y la seguridad del sistema mediante la estructuración lógica y física de la base de datos en PostgreSQL.

El diseño lógico y físico de la base de datos tiene un impacto directo en la eficiencia del sistema. Una base de datos bien estructurada permite gestionar grandes volúmenes de información sin comprometer el rendimiento. Por esta razón, en esta fase se enfatiza la optimización mediante índices, normalización y desnormalización estratégica, configuración avanzada de PostgreSQL, y la aplicación de mecanismos de seguridad robustos.

Diseño Lógico de la Base de Datos

El diseño lógico de la base de datos establece la estructura y relaciones entre las entidades de manera optimizada.

1 Refinamiento del Modelo Relacional

Para mejorar la eficiencia y la integridad de los datos, se realizaron los siguientes ajustes:

- **Vistas Materializadas**: Para optimizar consultas repetitivas en informes.
- **Estructura de Datos Optimizada**: Se eliminaron redundancias mediante normalización, pero también se aplicó desnormalización en vistas para mejorar la velocidad de consulta.
- **Triggers para Automatización**: Se agregaron disparadores para mantener consistencia en los datos.

Trigger para Actualización Automática de Estado del Equipo:

```
CREATE OR REPLACE FUNCTION actualizar_estado_equipo()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.fecha_fin < CURRENT_DATE THEN
        UPDATE equipo SET estado = 'disponible' WHERE equipo_id = NEW.equipo_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_actualizar_estado_equipo
AFTER UPDATE ON contrato_alquiler
FOR EACH ROW EXECUTE FUNCTION actualizar_estado_equipo();
```

2 Consideraciones de Escalabilidad y Rendimiento

- **Particionamiento de Tablas** en facturacion para acelerar consultas históricas.
- **Estrategias de Caché** con PostgreSQL para minimizar la carga en el servidor.

Particionamiento de Facturación por Año:

```
CREATE TABLE facturacion_2024 PARTITION OF facturacion
FOR VALUES FROM ('2024-01-01') TO ('2024-12-31');

CREATE TABLE facturacion_2025 PARTITION OF facturacion
FOR VALUES FROM ('2025-01-01') TO ('2025-12-31');
```

Diseño Físico de la Base de Datos

1 Elección de Tipos de Almacenamiento

- **Uso de JSONB en informes** para mayor flexibilidad.
- **Particionamiento en** facturacion para acelerar búsquedas por fechas.

2 Funciones Almacenadas para Optimización

Función para Obtener el Total de Alquileres por Cliente:

```
CREATE OR REPLACE FUNCTION obtener_total_alquileres(cliente_id INT)
RETURNS INT AS $$
DECLARE
    total INT;
BEGIN
    SELECT COUNT(*) INTO total FROM contrato_alquiler WHERE contrato_alquiler.cliente_id = cliente_id;
    RETURN total;
END;
$$ LANGUAGE plpgsql;
```

Función para Obtener el Costo Total de los Alquileres por Cliente:

```
CREATE OR REPLACE FUNCTION obtener_costo_total_alquiler(cliente_id INT)
RETURNS NUMERIC AS $$
DECLARE
    total NUMERIC;
BEGIN
    SELECT SUM(precio_alquiler) INTO total FROM contrato_alquiler WHERE contrato_alquiler.cliente_id =
cliente_id;
    RETURN total;
END;
$$ LANGUAGE plpgsql;
```

3 Configuración de Parámetros en PostgreSQL

Se realizaron optimizaciones en la configuración del servidor:

```
ALTER SYSTEM SET work_mem = '512MB';
ALTER SYSTEM SET maintenance_work_mem = '1GB';
ALTER SYSTEM SET effective_cache_size = '8GB';
```

Tuneado de Consultas

1 Análisis de Planes de Ejecución

El uso de EXPLAIN ANALYZE permitió optimizar consultas claves:

Vista Materializada para Contratos Activos:

```
CREATE MATERIALIZED VIEW vista_contratos_activos AS
SELECT cl.nombre, co.contrato_id, eq.tipo, co.fecha_inicio, co.fecha_fin
FROM cliente cl
JOIN contrato_alquiler co ON cl.cliente_id = co.cliente_id
JOIN equipo eq ON co.equipo_id = eq.equipo_id
WHERE co.fecha_fin >= CURRENT_DATE;

REFRESH MATERIALIZED VIEW vista_contratos_activos;
```

Consulta Optimizada con JOIN Indexado:

```
EXPLAIN ANALYZE
SELECT cl.nombre, co.contrato_id, eq.tipo, co.fecha_inicio, co.fecha_fin
FROM cliente cl
JOIN contrato_alquiler co ON cl.cliente_id = co.cliente_id
JOIN equipo eq ON co.equipo_id = eq.equipo_id
WHERE co.fecha_inicio >= '2024-01-01';
```

Estrategia de Backup y Recuperación

Para proteger los datos:

- **Backups Incrementales** diarios.
- **Backup Completo Semanal**.
- **Cifrado AES-256** en los backups.

Backup Completo con PostgreSQL:

pg_dump -U usuario -h localhost -F c -b -v -f backup_base_datos.backup base_datos

Restauración de Backup:

pg_restore -U usuario -h localhost -d base_datos -v backup_base_datos.backup

Automatización de Backup con Cron Job:

crontab -e

0 2 * * * pg_dump -U usuario -h localhost -F c -b -v -f /backups/backup_$(date +\%Y\%m\%d).backup base_datos

Conclusión

El diseño lógico y físico de la base de datos ha sido optimizado con técnicas avanzadas para garantizar rendimiento, escalabilidad y seguridad. Se implementaron triggers, funciones almacenadas, vistas materializadas y particionamiento para mejorar la eficiencia y automatización del sistema. Las optimizaciones aplicadas permiten manejar grandes volúmenes de información sin degradar la velocidad de acceso, asegurando que "Construcción Rentable" pueda operar con eficiencia y seguridad.