Resumen de Gestión y Modelado de Datos

Juan Carlos Martínez Arias

Mayo 2025

1. Álgebra Relacional

1.1. Operaciones Fundamentales

- Selección (σ): Extrae tuplas que cumplen una condición
- Proyección (π): Extrae columnas específicas
- Unión ($\cup$): Combina tuplas de dos relaciones
- Diferencia ($-$): Tuplas en R1 pero no en R2
- Producto Cartesiano ($\times$): Combina cada tupla de R1 con cada tupla de R2

1.2. Operaciones Derivadas

- Intersección ($\cap$): Tuplas comunes a ambas relaciones
- Join Natural ($\bowtie$): Une tablas por columnas con igual nombre
- Theta Join ($\bowtie_\theta$): Join con condición arbitraria
- Outer Joins:
  - Left Outer Join (): Todas las tuplas de R1 más las coincidentes de R2
  - Right Outer Join (): Todas las tuplas de R2 más las coincidentes de R1
  - Full Outer Join (): Todas las tuplas de R1 y R2

2. Normalización (Definiciones Formales)

2.1. Primera Forma Normal (1FN)

Definición formal: Una relación R está en 1FN si y solo si:

1. Todos los atributos son atómicos (indivisibles)
2. No existen grupos repetitivos (arrays o conjuntos en una celda)
3. Existe una clave primaria que identifica cada tupla de manera única

Propiedades:

- Elimina la redundancia de grupos repetitivos
- Establece la identificación única de registros
- Es el requisito mínimo para considerar una estructura como relación

2.2. Segunda Forma Normal (2FN)

Definición formal: Una relación R está en 2FN si:

1. Está en 1FN
2. Todo atributo no primo depende **completamente** de toda clave candidata (no de un subconjunto propio)

Matemáticamente: Para toda dependencia funcional $X \to Y$ donde Y es un atributo no primo:

$$X' \subset X \text{ tal que } X' \to Y \tag{1}$$

Propiedades:

- Elimina dependencias parciales de la clave primaria
- Reduce anomalías de actualización
- Requiere descomponer tablas con claves compuestas

2.3. Tercera Forma Normal (3FN)

Definición formal: Una relación R está en 3FN si:

1. Está en 2FN
2. No existen dependencias funcionales transitivas de atributos no primos a la clave

Matemáticamente: Para toda dependencia funcional $X \to Y$:

$$\text{O } X \text{ es superclave, o } Y \text{ es atributo primo} \tag{2}$$

Alternativamente (Definición de Date): R está en 3FN si para toda dependencia no trivial $X \to A$, se cumple que:

$$X \text{ es superclave o } A \text{ es miembro de alguna clave candidata} \tag{3}$$

Propiedades:

- Elimina dependencias transitivas
- Reduce redundancia entre atributos no clave
- Mantiene integridad referencial

2.4. Forma Normal de Boyce-Codd (FNBC)

Definición formal: Una relación R está en FNBC si para toda dependencia funcional no trivial $X \rightarrow Y$:

$$X \text{ es superclave de } R \tag{4}$$

Teorema: Toda relación en FNBC está automáticamente en 3FN, pero no viceversa.

Diferencia clave con 3FN:

- En 3FN se permiten dependencias donde Y es atributo primo
- En FNBC **siempre** el determinante debe ser superclave

Propiedades:

- Elimina todas las redundancias por dependencias funcionales
- Proporciona la máxima protección contra anomalías
- Puede perder algunas dependencias en la descomposición

2.5. Resumen de las Formas Normales

Cuadro 1: Jerarquía de Formas Normales

Forma Normal	**Problema que Resuelve**
1FN	Atomicidad y estructura básica
2FN	Dependencias parciales
3FN	Dependencias transitivas
FNBC	Dependencias donde el determinante no es superclave
4FN	Dependencias multivaluadas
5FN	Dependencias de unión

3. SQL (DDL y DML)

3.1. Ejemplos SELECT

```
-- Consulta con JOIN y filtros
SELECT H.ced_empleado, H.fecha_inicio, C.nom_cargo
```

```
FROM historia_cargos H
NATURAL JOIN CARGO C
WHERE H.ced_empleado = 1557235452;

-- Consulta con GROUP BY y HAVING
SELECT nom_cargo, AVG(salario) AS Prom_Sal_Cargo
FROM empleado NATURAL JOIN cargo
GROUP BY nom_cargo
HAVING AVG(salario) > 4000000;
```

3.2. Ejemplos UPDATE/DELETE/ALTER

```
-- Actualizar con subconsulta
UPDATE cliente
SET descuento = 0.10
WHERE codCiud IN (SELECT codCiud FROM ciudad WHERE nombre='Cali');

-- Eliminar registros
DELETE FROM compra
WHERE cc IN (SELECT cc FROM cliente WHERE nombre = 'Ana Torres');

-- Modificar tabla
ALTER TABLE prueba RENAME TO nuevaPrueba;
ALTER TABLE nuevaPrueba ADD PRIMARY KEY (col2, col3);
```

4. PL/SQL

4.1. Ejemplos

```
-- Bloque PL/SQL básico
DECLARE
    n_sales NUMBER := 2000000;
BEGIN
    IF n_sales > 100000 THEN
        DBMS_OUTPUT.PUT_LINE('Ventas mayores que 100K');
    END IF;
END;
```

4.2. Procedimiento Almacenado

```
CREATE OR REPLACE PROCEDURE PdcCompras(ccCliente NUMBER)
AS
    descuentocl NUMBER(10,2);
BEGIN
    SELECT SUM(precio*cantidad)*0.1 INTO descuentocl
    FROM compra
```

```
    WHERE cc = ccCliente AND
          fecha BETWEEN '01/09/2024' AND '31/12/2024';
    IF descuentocl IS NOT NULL THEN
        UPDATE cliente SET descuento=descuento+descuentocl
        WHERE cc= ccCliente;
    END IF;
END;
```

4.3. Función

```
CREATE OR REPLACE FUNCTION fdctoCompras(ccCliente NUMBER)
RETURN NUMBER
AS
    descuentocl NUMBER(10,2);
BEGIN
    SELECT SUM(precio*cantidad)*0.1 INTO descuentocl
    FROM compra
    WHERE cc = ccCliente AND
          fecha BETWEEN '01/09/2024' AND '31/12/2024';
    RETURN descuentocl;
END;
```

4.4. Trigger

```
CREATE OR REPLACE TRIGGER customers_audit_trg
AFTER UPDATE OR DELETE ON customers
FOR EACH ROW
DECLARE
    l_transaction VARCHAR2(10);
BEGIN
    l_transaction := CASE
        WHEN UPDATING THEN 'UPDATE'
        WHEN DELETING THEN 'DELETE'
    END;
    INSERT INTO audits (table_name, transaction_name, by_user, transaction_date)
    VALUES('CUSTOMERS', l_transaction, USER, SYSDATE);
END;
```

4.5. Cursor

```
DECLARE
    CURSOR c_product IS
        SELECT product_name, list_price
        FROM products
        ORDER BY list_price DESC;
BEGIN
```

```
    FOR r_product IN c_product LOOP
        dbms_output.put_line(r_product.product_name || ': $' || r_product.list_price);
    END LOOP;
END;
```

5. Tipos de JOIN en SQL

- **INNER JOIN**: Solo registros que coinciden en ambas tablas
- **LEFT JOIN**: Todos los registros de la tabla izquierda y coincidencias de la derecha
- **RIGHT JOIN**: Todos los registros de la tabla derecha y coincidencias de la izquierda
- **FULL JOIN**: Todos los registros de ambas tablas
- **CROSS JOIN**: Producto cartesiano de ambas tablas
- **NATURAL JOIN**: Join automático por columnas con igual nombre
- **SELF JOIN**: Join de una tabla consigo misma

5.1. Comparación: Bases Relacionales vs NoSQL

Cuadro 2: Comparación entre Bases Relacionales y NoSQL

Característica	**Relacional (SQL)**	**NoSQL (MongoDB)**
Modelo de datos	Tablas con esquema fijo	Documentos flexibles
Esquema	Definido previamente	Dinámico/sin esquema
Escalabilidad	Vertical	Horizontal
Transacciones	ACID completas	ACID limitadas
Lenguaje	SQL	API específica
Joins	Soporte completo	Referencias o datos embebidos
Consistencia	Fuerte	Eventual (configurable)

5.2. Ventajas de MongoDB

- **Flexibilidad**: Estructura de datos adaptable a cambios
- **Escalabilidad horizontal**: Distribución fácil en múltiples servidores
- **Rendimiento**: Optimizado para operaciones de lectura/escritura rápidas
- **Modelado para datos complejos**: Soporte nativo para arrays y objetos anidados
- **Alta disponibilidad**: Réplicas automáticas y tolerancia a fallos

5.3. Desventajas de MongoDB

- **Consistencia eventual**: Puede haber retrasos en la propagación de datos
- **Duplicación de datos**: Para evitar joins, se repite información
- **Transacciones limitadas**: Hasta hace poco no soportaba transacciones multi-documento
- **Madurez**: Menor madurez que sistemas relacionales (herramientas menos desarrolladas)
- **Estándares**: Falta de un lenguaje estandarizado como SQL

5.4. Casos de Uso Ideales para MongoDB

- Contenido gestionado y catálogos de productos
- Sistemas de recomendación y personalización
- Datos de IoT y series temporales
- Aplicaciones móviles y en tiempo real
- Big Data y análisis de datos no estructurados
- Sistemas donde el esquema cambia frecuentemente

5.5. Operaciones Básicas en MongoDB

```
// Insertar
db.usuarios.insertOne({
    nombre: "Ana Gómez",
    edad: 28,
    intereses: ["deportes", "música"]
});

// Consultar
db.usuarios.find({edad: {$gt: 25}});

// Actualizar
db.usuarios.updateOne(
    {nombre: "Ana Gómez"},
    {$set: {edad: 29}}
);

// Eliminar
db.usuarios.deleteOne({nombre: "Ana Gómez"});

// Agregación
```

```
db.ventas.aggregate([
    {$match: {fecha: {$gt: ISODate("2025-01-01")}}},
    {$group: {_id: "$producto", total: {$sum: "$cantidad"}}}
]);
```

5.6. Cuándo Elegir MongoDB vs SQL

- **Elegir SQL** cuando:
  - Los datos son altamente estructurados y estables
  - Se necesitan transacciones complejas
  - La consistencia fuerte es crítica
  - Hay muchas relaciones entre entidades
- **Elegir MongoDB** cuando:
  - Los datos son semiestructurados o no estructurados
  - El esquema evoluciona rápidamente
  - Se necesita escalar horizontalmente
  - El rendimiento en lectura/escritura es prioritario
  - Se trabaja con datos jerárquicos o anidados