# CAPÍTULO 2: LENGUAJE ESTRUCTURADO DE CONSULTAS (SQL)

## FUNDAMENTOS TEÓRICOS

---

## INTRODUCCIÓN AL SQL

### ¿Qué es SQL?

**SQL (Structured Query Language)** es el lenguaje estándar de acceso y manipulación de bases de datos relacionales. Fue desarrollado por IBM en los años 70 y posteriormente estandarizado por ANSI (American National Standards Institute) e ISO (International Organization for Standardization).

### Características Principales de SQL

1. **Lenguaje Declarativo**: El usuario especifica QUÉ quiere obtener, no CÓMO obtenerlo
2. **Lenguaje No Procedimental**: Se enfoca en el resultado, no en el proceso
3. **Estándar Internacional**: Funciona en múltiples SGBD con ligeras variaciones
4. **Potente y Flexible**: Desde consultas simples hasta operaciones complejas
5. **Integrable**: Puede embeberse en lenguajes de programación

### Categorías de SQL

SQL se divide en cinco sublenguajes principales:

#### 1. DDL (Data Definition Language)

Lenguaje de definición de datos. Define la estructura de la base de datos.

- Comandos: CREATE, ALTER, DROP, TRUNCATE

**2. DML (Data Manipulation Language)**

Lenguaje de manipulación de datos. Modifica los datos almacenados.

- Comandos: INSERT, UPDATE, DELETE

**3. DQL (Data Query Language)**

Lenguaje de consulta de datos. Recupera información.

- Comando: SELECT

**4. DCL (Data Control Language)**

Lenguaje de control de datos. Gestiona permisos y accesos.

- Comandos: GRANT, REVOKE

**5. TCL (Transaction Control Language)**

Lenguaje de control de transacciones. Gestiona la integridad transaccional.

- Comandos: COMMIT, ROLLBACK, SAVEPOINT

---

# 2.1 SQL DDL Y RESTRICCIONES DE INTEGRIDAD

## 2.1.1 Conceptos Fundamentales de DDL

**Data Definition Language (DDL)** es el conjunto de comandos SQL que permiten definir, modificar y eliminar la estructura de los objetos de la base de datos.

**Objetos de Base de Datos**

Los principales objetos que se gestionan con DDL son:

1. **Tablas**: Estructuras que almacenan datos en filas y columnas
2. **Índices**: Estructuras que aceleran las búsquedas
3. **Vistas**: Consultas almacenadas como tablas virtuales
4. **Secuencias**: Generadores de números únicos
5. **Sinónimos**: Alias para objetos
6. **Procedimientos y Funciones**: Código almacenado ejecutable

### 2.1.2 Tipos de Datos

Los tipos de datos definen qué clase de información puede almacenar cada columna.

**Tipos Numéricos**

**Enteros**:

- **INTEGER/INT**: Números enteros (típicamente 4 bytes)
- **SMALLINT**: Enteros pequeños (2 bytes)
- **BIGINT**: Enteros grandes (8 bytes)
- **TINYINT**: Enteros muy pequeños (1 byte) - MySQL, SQL Server

**Decimales**:

- **DECIMAL(p,s)/NUMERIC(p,s)**: Precisión fija (p=precisión total, s=decimales)
- **FLOAT**: Punto flotante de precisión simple
- **DOUBLE/DOUBLE PRECISION**: Punto flotante de doble precisión
- **REAL**: Punto flotante (implementación variable)

**Tipos de Cadena**

**Cadenas de caracteres**:

- **CHAR(n)**: Cadena de longitud fija
- **VARCHAR(n)**: Cadena de longitud variable
- **TEXT**: Texto largo sin límite específico

**Cadenas Unicode**:

- **NCHAR(n)**: Cadena Unicode de longitud fija (SQL Server)
- **NVARCHAR(n)**: Cadena Unicode variable (SQL Server)

**Binarios**:

- **BINARY(n)**: Datos binarios de longitud fija
- **VARBINARY(n)**: Datos binarios de longitud variable
- **BLOB**: Objetos binarios grandes (Binary Large Object)

**Tipos de Fecha y Hora**

- **DATE**: Solo fecha (YYYY-MM-DD)
- **TIME**: Solo hora (HH:MM:SS)
- **DATETIME**: Fecha y hora combinadas
- **TIMESTAMP**: Marca temporal con zona horaria
- **YEAR**: Solo año (MySQL)

**Tipos Especiales**

- **BOOLEAN**: Valores verdadero/falso (TRUE/FALSE)

- **UUID**: Identificador único universal (PostgreSQL)
- **JSON**: Almacenamiento de documentos JSON
- **XML**: Documentos XML
- **Tipos Geométricos**: POINT, POLYGON, etc. (PostgreSQL, MySQL)

### 2.1.3 Restricciones de Integridad

Las **restricciones de integridad** son reglas que garantizan la validez, consistencia y precisión de los datos.

**Tipos de Restricciones**

**1. Restricciones de Dominio**

Definen el conjunto de valores válidos para una columna.

- **NOT NULL**: La columna no puede contener valores nulos
- **DEFAULT**: Valor predeterminado si no se especifica
- **CHECK**: Condición que debe cumplir el valor

**2. Restricciones de Clave (Key Constraints)**

Identifican de manera única las filas de una tabla.

- **PRIMARY KEY (Clave Primaria)**:
  - Identifica únicamente cada fila
  - No puede contener valores NULL
  - Solo puede haber una por tabla
  - Puede ser simple (una columna) o compuesta (varias columnas)

- **UNIQUE (Unicidad)**:
  - Garantiza valores únicos en la columna
  - Puede aceptar valores NULL (excepto SQL Server)
  - Puede haber múltiples restricciones UNIQUE por tabla

**3. Restricciones de Integridad Referencial**

Mantienen la consistencia entre tablas relacionadas.

- **FOREIGN KEY (Clave Foránea)**:
  - Referencia a la clave primaria de otra tabla
  - Garantiza que los valores existan en la tabla referenciada
  - Controla las operaciones mediante:
    - **ON DELETE**: Qué hacer al eliminar el registro padre
    - **ON UPDATE**: Qué hacer al actualizar el registro padre

**Acciones Referenciales**:

- **CASCADE**: Propaga la operación a los registros hijos
- **SET NULL**: Establece NULL en los registros hijos
- **SET DEFAULT**: Establece el valor por defecto
- **RESTRICT/NO ACTION**: Impide la operación si hay hijos

**Niveles de Restricciones**

**Restricciones a Nivel de Columna**: Se definen junto con la columna individual.

**Restricciones a Nivel de Tabla**: Se definen después de todas las columnas, útiles para claves compuestas.

### 2.1.4 Comandos DDL Principales

**CREATE**

Crea nuevos objetos en la base de datos.

**Sintaxis CREATE TABLE**:

```sql
```

```
CREATE TABLE [IF NOT EXISTS] nombre_tabla (
    columna1 tipo_dato [restricciones_columna],
    columna2 tipo_dato [restricciones_columna],
    ...
    [restriccion_tabla1],
    [restriccion_tabla2],
    ...
) [opciones_tabla];

-- Restricciones de columna:
-- [NOT NULL]
-- [DEFAULT valor]
-- [UNIQUE]
-- [PRIMARY KEY]
-- [CHECK (condición)]
-- [REFERENCES tabla_ref(columna) [ON DELETE acción] [ON UPDATE acción]]

-- Restricciones de tabla:
-- [CONSTRAINT nombre] PRIMARY KEY (columna1, columna2, ...)
-- [CONSTRAINT nombre] FOREIGN KEY (columna1, ...)
--     REFERENCES tabla_ref(columna_ref1, ...)
--     [ON DELETE {CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION}]
--     [ON UPDATE {CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION}]
-- [CONSTRAINT nombre] UNIQUE (columna1, columna2, ...)
-- [CONSTRAINT nombre] CHECK (condición)
```

**Sintaxis CREATE INDEX**:

```
sql

CREATE [UNIQUE] INDEX nombre_indice
ON nombre_tabla (columna1 [ASC|DESC], columna2 [ASC|DESC], ...);
```

**Sintaxis CREATE VIEW**:

```sql
CREATE [OR REPLACE] VIEW nombre_vista [(columna1, columna2, ...)]
AS
   SELECT ...
[WITH CHECK OPTION];
```

**Sintaxis CREATE SEQUENCE** (Oracle, PostgreSQL):

```sql
CREATE SEQUENCE nombre_secuencia
   [START WITH valor_inicial]
   [INCREMENT BY incremento]
   [MINVALUE valor_minimo | NO MINVALUE]
   [MAXVALUE valor_maximo | NO MAXVALUE]
   [CYCLE | NO CYCLE]
   [CACHE cantidad | NO CACHE];
```

**ALTER**

Modifica objetos existentes.

**Sintaxis ALTER TABLE**:

```sql
```

```sql
-- Agregar columna
ALTER TABLE nombre_tabla
ADD [COLUMN] nombre_columna tipo_dato [restricciones];

-- Eliminar columna
ALTER TABLE nombre_tabla
DROP [COLUMN] nombre_columna [CASCADE | RESTRICT];

-- Modificar columna
ALTER TABLE nombre_tabla
MODIFY [COLUMN] nombre_columna nuevo_tipo_dato [restricciones];  -- Oracle, MySQL

ALTER TABLE nombre_tabla
ALTER [COLUMN] nombre_columna TYPE nuevo_tipo_dato;  -- PostgreSQL

ALTER TABLE nombre_tabla
ALTER COLUMN nombre_columna nuevo_tipo_dato;  -- SQL Server

-- Renombrar columna
ALTER TABLE nombre_tabla
RENAME COLUMN nombre_antiguo TO nombre_nuevo;  -- Oracle, PostgreSQL

-- Agregar restricción
ALTER TABLE nombre_tabla
ADD CONSTRAINT nombre_restriccion tipo_restriccion (columnas);

-- Eliminar restricción
ALTER TABLE nombre_tabla
DROP CONSTRAINT nombre_restriccion;

-- Renombrar tabla
```

```
ALTER TABLE nombre_tabla_antigua
RENAME TO nombre_tabla_nueva;  -- Oracle, PostgreSQL, MySQL
```

**DROP**

Elimina objetos de la base de datos.

**Sintaxis DROP**:

```
sql

-- Eliminar tabla
DROP TABLE [IF EXISTS] nombre_tabla [CASCADE | RESTRICT];

-- Eliminar índice
DROP INDEX [IF EXISTS] nombre_indice;

-- Eliminar vista
DROP VIEW [IF EXISTS] nombre_vista [CASCADE | RESTRICT];

-- Eliminar secuencia
DROP SEQUENCE [IF EXISTS] nombre_secuencia [CASCADE | RESTRICT];

-- Eliminar base de datos
DROP DATABASE [IF EXISTS] nombre_database;

-- Eliminar esquema
DROP SCHEMA [IF EXISTS] nombre_esquema [CASCADE | RESTRICT];
```

**Características**:

- Eliminación permanente
- No se puede deshacer (sin respaldo)

- **CASCADE**: Elimina objetos dependientes
- **RESTRICT**: Falla si existen objetos dependientes
- **IF EXISTS**: No genera error si el objeto no existe

#### TRUNCATE

Elimina todos los datos de una tabla pero mantiene su estructura.

**Sintaxis TRUNCATE**:

```sql
TRUNCATE TABLE nombre_tabla
[RESTART IDENTITY | CONTINUE IDENTITY]  -- PostgreSQL
[CASCADE | RESTRICT];  -- PostgreSQL
```

**Diferencias con DELETE**:

- Más rápido que DELETE
- No puede usarse con WHERE
- No activa triggers (generalmente)
- Reinicia secuencias/identidades (en algunos SGBD)
- No puede revertirse con ROLLBACK (en algunos SGBD)

### 2.1.5 Índices

Los **índices** son estructuras de datos que aceleran las operaciones de búsqueda.

#### Tipos de Índices

**Por Estructura**:

1. **B-Tree (Árbol B)**: Índice predeterminado en la mayoría de SGBD
2. **Hash**: Búsqueda por igualdad exacta
3. **Bitmap**: Para columnas con pocos valores distintos
4. **GiST/GIN**: Para tipos de datos complejos (PostgreSQL)

**Por Cobertura**:

1. **Índice Simple**: Sobre una sola columna
2. **Índice Compuesto**: Sobre múltiples columnas
3. **Índice de Cobertura**: Incluye todas las columnas necesarias para una consulta

**Por Unicidad**:

1. **Único (UNIQUE)**: No permite valores duplicados
2. **No Único**: Permite duplicados

**Ventajas de los Índices**

- Aceleran las búsquedas (SELECT)
- Mejoran el rendimiento de JOIN
- Optimizan ORDER BY y GROUP BY

**Desventajas de los Índices**

- Ocupan espacio en disco
- Ralentizan INSERT, UPDATE y DELETE
- Requieren mantenimiento

---

## 2.2 SQL DML (DATA MANIPULATION LANGUAGE)

### 2.2.1 Conceptos Fundamentales de DML

**Data Manipulation Language (DML)** es el conjunto de comandos que permiten insertar, modificar y eliminar datos en las tablas.

**Características de DML**

1. **Operaciones CRUD**: Create (INSERT), Read (SELECT), Update (UPDATE), Delete (DELETE)
2. **Transaccionales**: Pueden revertirse con ROLLBACK
3. **Afectan datos**: No modifican la estructura
4. **Registran cambios**: Generan entradas en logs de transacciones

### 2.2.2 Comando INSERT

Inserta nuevos registros en una tabla.

**Sintaxis INSERT - Forma 1 (valores explícitos)**:

```sql
INSERT INTO nombre_tabla [(columna1, columna2, ...)]
VALUES (valor1, valor2, ...);
```

**Sintaxis INSERT - Forma 2 (múltiples filas)**:

```sql
```

```
INSERT INTO nombre_tabla [(columna1, columna2, ...)]
VALUES
   (valor1_fila1, valor2_fila1, ...),
   (valor1_fila2, valor2_fila2, ...),
   (valor1_fila3, valor2_fila3, ...);
```

**Sintaxis INSERT - Forma 3 (desde SELECT)**:

```
sql

INSERT INTO nombre_tabla [(columna1, columna2, ...)]
SELECT columna1, columna2, ...
FROM otra_tabla
[WHERE condición];
```

**Sintaxis INSERT - MySQL (con manejo de duplicados)**:

```
sql

-- Ignora si existe
INSERT IGNORE INTO nombre_tabla (columnas)
VALUES (valores);

-- Actualiza si existe
INSERT INTO nombre_tabla (columna1, columna2, columna3)
VALUES (valor1, valor2, valor3)
ON DUPLICATE KEY UPDATE
   columna2 = valor2_nuevo,
   columna3 = valor3_nuevo;
```

**Sintaxis INSERT - PostgreSQL (RETURNING)**:

```sql
INSERT INTO nombre_tabla (columnas)
VALUES (valores)
RETURNING *;  -- Devuelve el registro insertado
```

**Consideraciones**

- Las columnas NOT NULL deben tener valor
- Las columnas con DEFAULT pueden omitirse
- Las claves primarias deben ser únicas
- Las claves foráneas deben existir en la tabla referenciada

### 2.2.3 Comando UPDATE

Modifica datos existentes en una tabla.

**Sintaxis UPDATE - Forma básica**:

```sql
UPDATE nombre_tabla
 SET columna1 = valor1,
    columna2 = valor2,
    ...
[WHERE condición];
```

**Sintaxis UPDATE - Con subconsulta**:

```sql
```

```
UPDATE nombre_tabla
SET columna1 = (SELECT expresión FROM otra_tabla WHERE condición),
    columna2 = valor2
[WHERE condición];
```

**Sintaxis UPDATE - Con JOIN (SQL Server, MySQL)**:

```sql
-- SQL Server
UPDATE t1
SET t1.columna1 = t2.columna1,
    t1.columna2 = valor
FROM tabla1 t1
INNER JOIN tabla2 t2 ON t1.id = t2.id
WHERE condición;

-- MySQL
UPDATE tabla1 t1
INNER JOIN tabla2 t2 ON t1.id = t2.id
SET t1.columna1 = t2.columna1,
    t1.columna2 = valor
WHERE condición;
```

**Sintaxis UPDATE - PostgreSQL (FROM)**:

```sql
```

```
UPDATE tabla1 t1
SET columna1 = t2.columna1,
    columna2 = valor
FROM tabla2 t2
WHERE t1.id = t2.id
  AND condición;
```

**Precauciones**

- **Sin WHERE se modifican TODAS las filas ⚠**
- Validar restricciones antes de actualizar
- Considerar el impacto en claves foráneas
- Usar transacciones para cambios importantes

### 2.2.4 Comando DELETE

Elimina registros de una tabla.

**Sintaxis DELETE - Forma básica**:

```
sql

DELETE FROM nombre_tabla
[WHERE condición];
```

**Sintaxis DELETE - Con subconsulta**:

```
sql
```

```sql
DELETE FROM nombre_tabla
WHERE columna IN (
    SELECT columna
    FROM otra_tabla
    WHERE condición
);
```

**Sintaxis DELETE - Con JOIN (SQL Server, MySQL)**:

```sql
-- SQL Server
DELETE t1
FROM tabla1 t1
INNER JOIN tabla2 t2 ON t1.id = t2.id
WHERE condición;

-- MySQL
DELETE t1
FROM tabla1 t1
INNER JOIN tabla2 t2 ON t1.id = t2.id
WHERE condición;
```

**Sintaxis DELETE - PostgreSQL (USING)**:

```sql
DELETE FROM tabla1 t1
USING tabla2 t2
WHERE t1.id = t2.id
  AND condición;
```

**Diferencias DELETE vs TRUNCATE**

| Aspecto | DELETE | TRUNCATE |
|---|---|---|
| Velocidad | Lento (fila por fila) | Rápido |
| WHERE | Sí | No |
| Triggers | Sí se activan | Generalmente no |
| Rollback | Sí | Depende del SGBD |
| Logs | Registra cada fila | Mínimo registro |
| Identidad | No reinicia | Reinicia |

### 2.2.5 Transacciones y DML

Las operaciones DML son **transaccionales**, lo que significa que:

1. **Atomicidad**: Todo o nada
2. **Consistencia**: La BD pasa de un estado válido a otro válido
3. **Aislamiento**: Las transacciones no interfieren entre sí
4. **Durabilidad**: Los cambios confirmados son permanentes

**Control de Transacciones**

- **BEGIN/START TRANSACTION**: Inicia transacción explícita
- **COMMIT**: Confirma los cambios
- **ROLLBACK**: Revierte los cambios
- **SAVEPOINT**: Crea punto de restauración parcial

---

## 2.3 SQL SELECT (DATA QUERY LANGUAGE)

### 2.3.1 Fundamentos de SELECT

**SELECT** es el comando más utilizado en SQL. Recupera datos de una o más tablas.

#### Estructura Básica

**Sintaxis completa de SELECT**:

```sql
SELECT [ALL | DISTINCT]
    columna1 [AS alias1],
    columna2 [AS alias2],
    expresión [AS alias],
    función_agregación(columna) [AS alias]
FROM tabla1 [AS alias_tabla1]
    [JOIN tabla2 ON condición]
[WHERE condición]
[GROUP BY columna1, columna2, ...]
[HAVING condición_grupo]
[ORDER BY columna1 [ASC | DESC], ...]
[LIMIT número [OFFSET desplazamiento]];
```

#### Orden de Ejecución Lógico

1. **FROM / JOIN**: Identifica y combina tablas
2. **WHERE**: Filtra filas individuales
3. **GROUP BY**: Agrupa filas
4. **HAVING**: Filtra grupos
5. **SELECT**: Selecciona y calcula columnas

6. **DISTINCT**: Elimina duplicados
7. **ORDER BY**: Ordena resultados
8. **LIMIT / OFFSET**: Limita resultados

### 2.3.2 Operaciones de Selección

**Proyección**

**Definición**: Selección de columnas específicas de una tabla.

**Características**:

- Especifica qué columnas mostrar
- Puede incluir expresiones calculadas
- Puede usar alias para renombrar
- El * selecciona todas las columnas

**Selección (WHERE)**

**Definición**: Filtrado de filas según criterios.

**Operadores de Comparación**:

- `=`, `<>`, `!=`, `>`, `<`, `>=`, `<=`
- `BETWEEN valor1 AND valor2`
- `IN (lista_valores)`
- `LIKE 'patrón'`
- `IS NULL`, `IS NOT NULL`

**Operadores Lógicos**:

- `AND`: Todas las condiciones deben cumplirse
- `OR`: Al menos una condición debe cumplirse
- `NOT`: Niega una condición

**Comodines en LIKE**:

- `%`: Representa cero o más caracteres
- `_`: Representa exactamente un carácter

**DISTINCT**

Elimina filas duplicadas del resultado.

### 2.3.3 Ordenamiento (ORDER BY)

Organiza los resultados según una o más columnas.

**Sintaxis**:

```sql
ORDER BY columna1 [ASC | DESC] [NULLS FIRST | NULLS LAST],
    columna2 [ASC | DESC], ...
```

**Características**:

- **ASC**: Orden ascendente (predeterminado)
- **DESC**: Orden descendente
- Puede ordenar por múltiples columnas
- Puede ordenar por expresiones

### 2.3.4 Operaciones de JOIN

**JOIN** combina filas de dos o más tablas basándose en una condición relacionada.

**Sintaxis general**:

```sql
SELECT columnas
FROM tabla1 [AS alias1]
[INNER | LEFT | RIGHT | FULL | CROSS] JOIN tabla2 [AS alias2]
   ON condición_join
[WHERE condición];
```

**INNER JOIN**:

```sql
SELECT t1.columna1, t2.columna2
FROM tabla1 t1
INNER JOIN tabla2 t2 ON t1.id = t2.tabla1_id;
```

**LEFT JOIN**:

```sql
SELECT t1.columna1, t2.columna2
FROM tabla1 t1
LEFT JOIN tabla2 t2 ON t1.id = t2.tabla1_id;
```

**RIGHT JOIN**:

```sql
```

```sql
SELECT t1.columna1, t2.columna2
FROM tabla1 t1
RIGHT JOIN tabla2 t2 ON t1.id = t2.tabla1_id;
```

**FULL JOIN**:

```sql
SELECT t1.columna1, t2.columna2
FROM tabla1 t1
FULL OUTER JOIN tabla2 t2 ON t1.id = t2.tabla1_id;
```

**CROSS JOIN**:

```sql
SELECT t1.columna1, t2.columna2
FROM tabla1 t1
CROSS JOIN tabla2 t2;
```

**SELF JOIN**:

```sql
SELECT e.nombre AS empleado, j.nombre AS jefe
FROM empleados e
INNER JOIN empleados j ON e.jefe_id = j.empleado_id;
```

**Tipos de JOIN**

1. **INNER JOIN**: Solo coincidencias en ambas tablas

2. **LEFT JOIN**: Todas las filas de la izquierda + coincidencias
3. **RIGHT JOIN**: Todas las filas de la derecha + coincidencias
4. **FULL JOIN**: Todas las filas de ambas tablas
5. **CROSS JOIN**: Producto cartesiano
6. **SELF JOIN**: Tabla unida consigo misma

### 2.3.5 Funciones de Agregación

**Sintaxis de funciones de agregación**:

```sql
SELECT
    COUNT(*) AS total_filas,
    COUNT(columna) AS no_nulos,
    COUNT(DISTINCT columna) AS valores_unicos,
    SUM(columna_numerica) AS suma,
    AVG(columna_numerica) AS promedio,
    MIN(columna) AS minimo,
    MAX(columna) AS maximo
FROM tabla
[WHERE condición];
```

**Funciones Principales**

- **COUNT()**: Cuenta filas
- **SUM()**: Suma valores
- **AVG()**: Calcula promedio
- **MIN()**: Encuentra mínimo

- **MAX()**: Encuentra máximo
- **STDDEV()**: Desviación estándar
- **VARIANCE()**: Varianza

### 2.3.6 Agrupamiento (GROUP BY)

Agrupa filas que tienen valores idénticos.

**Sintaxis GROUP BY**:

```sql
SELECT
    columna1,
    columna2,
    función_agregación(columna3) AS alias
FROM tabla
[WHERE condición]
GROUP BY columna1, columna2
[HAVING condición_grupo]
[ORDER BY columna1];
```

**GROUP BY con ROLLUP** (MySQL, SQL Server):

```sql
SELECT departamento, cargo, SUM(salario)
FROM empleados
GROUP BY departamento, cargo WITH ROLLUP;
```

**HAVING**

Filtra grupos después del agrupamiento.

**Sintaxis**:

```sql
SELECT columna1, COUNT(*)
FROM tabla
GROUP BY columna1
HAVING COUNT(*) > 10;
```

**Diferencias WHERE vs HAVING**:

| Aspecto | WHERE | HAVING |
|---|---|---|
| Aplicación | Antes de agrupar | Después de agrupar |
| Funciones agregación | No permitidas | Permitidas |
| Filas afectadas | Individuales | Grupos |

## 2.3.7 Subconsultas (Subqueries)

Una **subconsulta** es una consulta SELECT dentro de otra consulta.

**Sintaxis general**:

```sql
```

```sql
-- En WHERE
SELECT columnas
FROM tabla
WHERE columna operador (SELECT columna FROM otra_tabla);

-- En FROM
SELECT * FROM (SELECT columnas FROM tabla) AS t;

-- En SELECT
SELECT columna1, (SELECT MAX(col) FROM tabla2) AS max_valor
FROM tabla1;
```

**Subconsulta escalar**:

```sql
SELECT nombre, salario
FROM empleados
WHERE salario > (SELECT AVG(salario) FROM empleados);
```

**Subconsulta correlacionada**:

```sql
SELECT e1.nombre, e1.salario
FROM empleados e1
WHERE e1.salario > (
    SELECT AVG(e2.salario)
    FROM empleados e2
    WHERE e2.departamento_id = e1.departamento_id
);
```

**Operadores con Subconsultas**

**IN**:

```sql
SELECT nombre FROM empleados
WHERE departamento_id IN (SELECT id FROM departamentos WHERE ciudad = 'Madrid');
```

**EXISTS**:

```sql
SELECT nombre FROM empleados e
WHERE EXISTS (SELECT 1 FROM ventas v WHERE v.vendedor_id = e.id);
```

**ANY / SOME**:

```sql
SELECT nombre, salario FROM empleados
WHERE salario > ANY (SELECT salario FROM empleados WHERE departamento_id = 10);
```

**ALL**:

```sql
SELECT nombre, salario FROM empleados
WHERE salario > ALL (SELECT salario FROM empleados WHERE departamento_id = 10);
```

## 2.3.8 Operadores de Conjunto

**UNION** (sin duplicados):

```sql
SELECT columna1, columna2 FROM tabla1
UNION
SELECT columna1, columna2 FROM tabla2;
```

**UNION ALL** (con duplicados):

```sql
SELECT columna1, columna2 FROM tabla1
UNION ALL
SELECT columna1, columna2 FROM tabla2;
```

**INTERSECT** (solo coincidencias):

```sql
SELECT columna1, columna2 FROM tabla1
INTERSECT
SELECT columna1, columna2 FROM tabla2;
```

**EXCEPT / MINUS** (diferencia):

```sql
SELECT columna1, columna2 FROM tabla1
EXCEPT
SELECT columna1, columna2 FROM tabla2;
```

### 2.3.9 Expresiones de Tabla Comunes (CTE)

**Sintaxis básica de CTE**:

```sql
WITH nombre_cte AS (
    SELECT columnas
    FROM tabla
    WHERE condición
)
SELECT * FROM nombre_cte;
```

**CTE múltiples**:

```sql
WITH
cte1 AS (SELECT * FROM tabla1),
cte2 AS (SELECT * FROM tabla2)
SELECT * FROM cte1 JOIN cte2 ON cte1.id = cte2.id;
```

**CTE recursiva**:

```sql
```

```sql
WITH RECURSIVE jerarquia AS (
  -- Ancla
  SELECT id, nombre, jefe_id, 1 AS nivel
  FROM empleados WHERE jefe_id IS NULL

  UNION ALL

  -- Recursión
  SELECT e.id, e.nombre, e.jefe_id, j.nivel + 1
  FROM empleados e
  JOIN jerarquia j ON e.jefe_id = j.id
)
SELECT * FROM jerarquia;
```

### 2.3.10 Funciones de Ventana

**Sintaxis general**:

```sql
función(columna) OVER (
  [PARTITION BY columna_particion]
  [ORDER BY columna_orden]
  [ROWS | RANGE especificación_marco]
)
```

**Funciones de ranking**:

```sql
```

```
SELECT
    nombre,
    salario,
    ROW_NUMBER() OVER (ORDER BY salario DESC) AS num_fila,
    RANK() OVER (ORDER BY salario DESC) AS ranking,
    DENSE_RANK() OVER (ORDER BY salario DESC) AS ranking_denso
FROM empleados;
```

**Funciones de valor**:

```
sql

SELECT
    fecha,
    ventas,
    LAG(ventas, 1) OVER (ORDER BY fecha) AS dia_anterior,
    LEAD(ventas, 1) OVER (ORDER BY fecha) AS dia_siguiente
FROM ventas_diarias;
```

**Marcos de ventana**:

```
sql

SELECT
    fecha,
    monto,
    SUM(monto) OVER (
        ORDER BY fecha
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS suma_acumulativa
FROM ventas;
```

---

## 2.4 FUNCIONES DEL SGBD

### 2.4.1 Concepto de Funciones

Las **funciones del SGBD** son operaciones predefinidas que realizan cálculos o transformaciones sobre datos.

**Clasificación General**

1. **Funciones Escalares**: Operan sobre un valor, devuelven un valor
2. **Funciones de Agregación**: Operan sobre múltiples valores, devuelven uno
3. **Funciones de Ventana**: Operan sobre un conjunto sin colapsar

### 2.4.2 Variables

Las **variables** son contenedores temporales de datos.

**Tipos de Variables**

**1. Variables de Usuario**: Definidas por el usuario, alcance de sesión **2. Variables de Sistema**: Predefinidas por el SGBD **3. Variables de Sesión**: Específicas de cada conexión

**Sintaxis por SGBD**:

**Oracle (PL/SQL)**:

```sql
```

```
DECLARE
   v_nombre VARCHAR2(50);
   v_edad NUMBER := 25;
BEGIN
   v_nombre := 'Juan';
END;
```

**SQL Server (T-SQL)**:

```
sql

DECLARE @nombre VARCHAR(50);
DECLARE @edad INT = 25;
SET @nombre = 'Juan';
```

**PostgreSQL (PL/pgSQL)**:

```
sql

DO $$
DECLARE
   v_nombre VARCHAR(50);
   v_edad INTEGER := 25;
BEGIN
   v_nombre := 'Juan';
END $$;
```

**MySQL**:

```
sql
```

```
SET @nombre = 'Juan';
SET @edad = 25;
```

### 2.4.3 Estructuras Condicionales

#### CASE Expression

#### CASE simple:

```
sql

SELECT nombre,
   CASE departamento
      WHEN 'Ventas' THEN 'V'
      WHEN 'IT' THEN 'T'
      ELSE 'O'
   END AS codigo
FROM empleados;
```

#### CASE evaluado:

```
sql

SELECT nombre, edad,
   CASE
      WHEN edad < 25 THEN 'Junior'
      WHEN edad BETWEEN 25 AND 40 THEN 'Semi-Senior'
      ELSE 'Senior'
   END AS categoria
FROM empleados;
```

#### IF-THEN-ELSE (Lenguajes Procedurales)

**Oracle**:

```sql
IF condición THEN
    instrucciones;
ELSIF otra_condición THEN
    otras_instrucciones;
ELSE
    instrucciones_finales;
END IF;
```

**SQL Server**:

```sql
IF condición
BEGIN
    instrucciones;
END
ELSE IF otra_condición
BEGIN
    otras_instrucciones;
END
```

### 2.4.4 Funciones Matemáticas

**Funciones básicas**:

```sql
```

```
ABS(n)          -- Valor absoluto
ROUND(n, d)     -- Redondeo
CEILING(n)      -- Redondeo hacia arriba
FLOOR(n)        -- Redondeo hacia abajo
TRUNC(n, d)     -- Truncar
MOD(n, m)       -- Módulo (resto)
POWER(n, m)     -- Potencia
SQRT(n)         -- Raíz cuadrada
SIGN(n)         -- Signo (-1, 0, 1)
```

**Funciones trigonométricas**:

```
sql

SIN(n), COS(n), TAN(n)
ASIN(n), ACOS(n), ATAN(n)
PI()            -- Constante π
```

### 2.4.5 Funciones de Cadenas

**Transformación**:

```
sql

UPPER(str)      -- A mayúsculas
LOWER(str)      -- A minúsculas
INITCAP(str)    -- Primera letra mayúscula
TRIM(str)       -- Quitar espacios
LTRIM(str)      -- Quitar espacios izquierda
RTRIM(str)      -- Quitar espacios derecha
LPAD(str, len, pad) -- Rellenar izquierda
RPAD(str, len, pad) -- Rellenar derecha
```

**Extracción**:

```sql
SUBSTRING(str, pos, len)  -- Extraer subcadena
LEFT(str, len)            -- Primeros n caracteres
RIGHT(str, len)           -- Últimos n caracteres
LENGTH(str)               -- Longitud
```

**Búsqueda y modificación**:

```sql
INSTR(str, substr)        -- Buscar posición
REPLACE(str, old, new)    -- Reemplazar
CONCAT(str1, str2, ...)   -- Concatenar
REVERSE(str)              -- Invertir
```

### 2.4.6 Funciones de Fecha y Hora

**Obtener fecha/hora actual**:

```sql
SYSDATE             -- Oracle
NOW()               -- MySQL, PostgreSQL
GETDATE()           -- SQL Server
CURRENT_DATE        -- Estándar SQL (solo fecha)
CURRENT_TIMESTAMP   -- Estándar SQL
```

**Extraer componentes**:

```sql
```

```
EXTRACT(YEAR FROM fecha)
YEAR(fecha), MONTH(fecha), DAY(fecha)
HOUR(time), MINUTE(time), SECOND(time)
QUARTER(fecha)      -- Trimestre
DAYOFWEEK(fecha)    -- Día de la semana
```

**Aritmética de fechas**:

```
sql

-- MySQL
DATE_ADD(fecha, INTERVAL 1 DAY)
DATE_SUB(fecha, INTERVAL 7 DAY)
DATEDIFF(fecha1, fecha2)

-- PostgreSQL
fecha + INTERVAL '1 day'
fecha - INTERVAL '7 days'
AGE(fecha1, fecha2)

-- SQL Server
DATEADD(DAY, 1, fecha)
DATEDIFF(DAY, fecha1, fecha2)
```

**Formateo**:

```
sql

TO_CHAR(fecha, 'DD/MM/YYYY')      -- Oracle, PostgreSQL
DATE_FORMAT(fecha, '%d/%m/%Y')    -- MySQL
FORMAT(fecha, 'dd/MM/yyyy')       -- SQL Server
```

### 2.4.7 Funciones de Conversión

**Conversión explícita**:

```sql
CAST(valor AS tipo)          -- Estándar SQL
CONVERT(tipo, valor)         -- SQL Server, MySQL
TO_NUMBER(str)               -- Oracle
TO_DATE(str, formato)        -- Oracle
TO_CHAR(valor)               -- Oracle
```

### 2.4.8 Funciones de NULL

**Manejo de valores nulos**:

```sql
COALESCE(val1, val2, ...)    -- Primer valor no NULL
NVL(val1, val2)              -- Oracle
ISNULL(val1, val2)           -- SQL Server
IFNULL(val1, val2)           -- MySQL
NULLIF(val1, val2)           -- NULL si son iguales
```

### 2.4.9 Funciones de Sistema

**Información del usuario/sesión**:

```sql
```

```
USER, CURRENT_USER          -- Usuario actual
SESSION_USER                -- Usuario de sesión
VERSION()                   -- Versión del SGBD
DATABASE()                  -- Base de datos actual
CONNECTION_ID()             -- ID de conexión
```

---

## 2.5 SQL DCL (DATA CONTROL LANGUAGE)

### 2.5.1 Conceptos Fundamentales de DCL

**Data Control Language (DCL)** controla el acceso a los datos y objetos.

#### Objetivos de DCL

1. **Seguridad**: Proteger datos sensibles
2. **Control de Acceso**: Regular quién puede hacer qué
3. **Auditoría**: Rastrear acciones
4. **Cumplimiento**: Satisfacer requisitos regulatorios

#### Principios de Seguridad

1. **Principio de Mínimo Privilegio**: Solo permisos necesarios
2. **Separación de Responsabilidades**: Dividir funciones críticas
3. **Defensa en Profundidad**: Múltiples capas de seguridad
4. **Auditoría Continua**: Monitorear accesos

### 2.5.2 Usuarios

Un **usuario** es una identidad que puede conectarse a la base de datos.

**Sintaxis CREATE USER**:

**Oracle**:

```sql
CREATE USER nombre_usuario
IDENTIFIED BY contraseña
DEFAULT TABLESPACE users
TEMPORARY TABLESPACE temp
QUOTA 100M ON users
PASSWORD EXPIRE
ACCOUNT UNLOCK;
```

**SQL Server**:

```sql
-- Crear LOGIN a nivel servidor
CREATE LOGIN nombre_login
WITH PASSWORD = 'Pass123$',
    DEFAULT_DATABASE = empresa;

-- Crear USER en base de datos
USE empresa;
CREATE USER nombre_usuario
FOR LOGIN nombre_login
WITH DEFAULT_SCHEMA = dbo;
```

**PostgreSQL**:

```sql
```

```
CREATE USER nombre_usuario
WITH PASSWORD 'Pass123$'
    LOGIN
    CREATEDB
    VALID UNTIL '2025-12-31'
    CONNECTION LIMIT 10;
```

**MySQL**:

```
sql

CREATE USER 'nombre_usuario'@'localhost'
IDENTIFIED BY 'Pass123$'
PASSWORD EXPIRE INTERVAL 90 DAY
ACCOUNT UNLOCK;
```

**Sintaxis ALTER USER**:

```
sql

-- Cambiar contraseña
ALTER USER nombre_usuario IDENTIFIED BY 'NuevaPass';

-- Bloquear cuenta
ALTER USER nombre_usuario ACCOUNT LOCK;

-- Cambiar fecha expiración
ALTER USER nombre_usuario VALID UNTIL '2026-12-31';
```

**Sintaxis DROP USER**:

```
sql
```

```
DROP USER nombre_usuario [CASCADE];
```

**Componentes de un Usuario**

1. **Nombre**: Identificador único
2. **Contraseña**: Credencial de autenticación
3. **Propiedades**: Límites, opciones
4. **Privilegios**: Permisos asignados

**Tipos de Autenticación**

1. **Base de Datos**: Credenciales en el SGBD
2. **Externa**: Sistema operativo, LDAP, AD
3. **Multifactor**: Combinación de métodos

### 2.5.3 Roles

Un **rol** es un conjunto de privilegios asignable a usuarios.

**Sintaxis CREATE ROLE**:

**Oracle**:

```
sql

CREATE ROLE desarrollador;
CREATE ROLE admin_rol IDENTIFIED BY admin_pass;
```

**SQL Server**:

```sql
CREATE ROLE desarrollador;
```

**PostgreSQL**:

```sql
CREATE ROLE desarrollador;
CREATE ROLE app_role NOLOGIN;
```

**MySQL (8.0+)**:

```sql
CREATE ROLE 'desarrollador';
CREATE ROLE 'analista', 'gerente';
```

**Asignar privilegios a roles**:

```sql
GRANT SELECT, INSERT, UPDATE, DELETE ON tabla TO desarrollador;
GRANT EXECUTE ON procedimiento TO desarrollador;
```

**Asignar roles a usuarios**:

```sql
```

```
-- Oracle
GRANT desarrollador TO juan_dev;

-- SQL Server
ALTER ROLE desarrollador ADD MEMBER juan_dev;

-- PostgreSQL
GRANT desarrollador TO juan_dev;

-- MySQL
GRANT 'desarrollador' TO 'juan_dev'@'localhost';
```

**Sintaxis DROP ROLE**:

```
sql

DROP ROLE nombre_rol;
```

**Ventajas de los Roles**

1. **Simplificación**: Gestión centralizada
2. **Consistencia**: Permisos uniformes
3. **Mantenibilidad**: Cambios en un lugar
4. **Escalabilidad**: Fácil agregar usuarios

**Roles Predefinidos**

- **Oracle**: CONNECT, RESOURCE, DBA
- **SQL Server**: db_owner, db_datareader, db_datawriter
- **PostgreSQL**: pg_read_all_data, pg_write_all_data

### 2.5.4 Privilegios

Los **privilegios** son permisos específicos.

**Categorías**

**1. Privilegios de Sistema**:

- CREATE SESSION, CREATE TABLE, CREATE USER, etc.

**2. Privilegios sobre Objetos**:

- SELECT, INSERT, UPDATE, DELETE, EXECUTE, etc.

**Granularidad**

1. **Global**: Todo el servidor
2. **Base de Datos**: Una BD completa
3. **Esquema**: Todos los objetos de un esquema
4. **Objeto**: Un objeto específico
5. **Columna**: Columnas específicas

### 2.5.5 Comando GRANT

**Sintaxis general**:

```sql
GRANT privilegio [(columnas)]
ON objeto
TO {usuario | rol | PUBLIC}
[WITH GRANT OPTION | WITH ADMIN OPTION];
```

**Ejemplos**:

**Privilegios sobre objetos**:

```sql
-- Privilegio simple
GRANT SELECT ON tabla TO usuario;

-- Múltiples privilegios
GRANT SELECT, INSERT, UPDATE, DELETE ON tabla TO usuario;

-- Todos los privilegios
GRANT ALL PRIVILEGES ON tabla TO usuario;

-- Privilegios sobre columnas
GRANT UPDATE (columna1, columna2) ON tabla TO usuario;

-- Con capacidad de otorgar a otros
GRANT SELECT ON tabla TO usuario WITH GRANT OPTION;
```

**Privilegios de sistema (Oracle)**:

```sql
GRANT CREATE SESSION TO usuario;
GRANT CREATE TABLE TO usuario;
```

**Asignar roles**:

```sql
```

```
GRANT nombre_rol TO usuario;
GRANT rol1, rol2 TO usuario;
```

**Sobre esquemas (PostgreSQL)**:

```
sql

GRANT USAGE ON SCHEMA esquema TO usuario;
GRANT CREATE ON SCHEMA esquema TO usuario;
GRANT ALL PRIVILEGES ON SCHEMA esquema TO usuario;

-- Sobre todas las tablas del esquema
GRANT SELECT, INSERT, UPDATE, DELETE
ON ALL TABLES IN SCHEMA esquema TO usuario;
```

**Sobre base de datos (MySQL)**:

```
sql

GRANT ALL PRIVILEGES ON db.* TO 'usuario'@'host';
GRANT SELECT, INSERT ON db.* TO 'usuario'@'host';
```

### 2.5.6 Comando REVOKE

**Sintaxis general**:

```
sql

REVOKE privilegio [(columnas)]
ON objeto
FROM {usuario | rol | PUBLIC}
[CASCADE | RESTRICT];
```

**Ejemplos**:

```sql
-- Revocar privilegio simple
REVOKE SELECT ON tabla FROM usuario;

-- Revocar múltiples
REVOKE SELECT, INSERT, UPDATE FROM usuario;

-- Revocar todos
REVOKE ALL PRIVILEGES ON tabla FROM usuario;

-- Con CASCADE
REVOKE SELECT ON tabla FROM usuario CASCADE;

-- Revocar solo GRANT OPTION
REVOKE GRANT OPTION FOR SELECT ON tabla FROM usuario;
```

**Opciones**:

- **CASCADE**: Revoca privilegios otorgados en cascada
- **RESTRICT**: Falla si hay dependencias

### 2.5.7 Esquemas

Un **esquema** es un contenedor lógico de objetos.

**Sintaxis CREATE SCHEMA**:

**Oracle** (Esquema = Usuario):

```sql
```

```
CREATE USER nombre_esquema IDENTIFIED BY password;
```

**SQL Server**:

```
sql

CREATE SCHEMA nombre_esquema AUTHORIZATION propietario;
```

**PostgreSQL**:

```
sql

CREATE SCHEMA nombre_esquema;
CREATE SCHEMA IF NOT EXISTS nombre_esquema;
```

**MySQL** (Schema = Database):

```
sql

CREATE SCHEMA nombre_esquema;
CREATE DATABASE nombre_esquema;
```

**Sintaxis ALTER SCHEMA**:

```
sql

-- SQL Server: Cambiar propietario
ALTER AUTHORIZATION ON SCHEMA::nombre TO nuevo_propietario;

-- PostgreSQL: Renombrar
ALTER SCHEMA nombre_antiguo RENAME TO nombre_nuevo;
```

**Sintaxis DROP SCHEMA**:

```sql
DROP SCHEMA nombre_esquema;
DROP SCHEMA nombre_esquema CASCADE;
```

**Propósito**

1. **Organización**: Agrupar objetos
2. **Seguridad**: Asignar permisos por esquema
3. **Separación**: Aislar datos
4. **Naming**: Evitar conflictos

## 2.5.8 Permisos sobre Esquemas

**PostgreSQL**:

```sql
GRANT USAGE ON SCHEMA esquema TO usuario;
GRANT CREATE ON SCHEMA esquema TO usuario;
GRANT ALL PRIVILEGES ON SCHEMA esquema TO usuario;

-- Permisos sobre objetos del esquema
GRANT SELECT, INSERT ON ALL TABLES IN SCHEMA esquema TO usuario;

-- Permisos por defecto para objetos futuros
ALTER DEFAULT PRIVILEGES IN SCHEMA esquema
GRANT SELECT, INSERT ON TABLES TO usuario;
```

**SQL Server**:

```sql
GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::esquema TO usuario;
GRANT EXECUTE ON SCHEMA::esquema TO usuario;
GRANT CONTROL ON SCHEMA::esquema TO usuario;
```

### 2.5.9 Mejores Prácticas de Seguridad

#### 1. Gestión de Contraseñas

- Contraseñas fuertes y complejas
- Cambiar periódicamente
- No compartir credenciales

#### 2. Principio de Mínimo Privilegio

- Otorgar solo permisos necesarios
- Revisar y ajustar periódicamente

#### 3. Separación de Ambientes

- Desarrollo, Pruebas, Producción separados
- Permisos más restrictivos en producción

#### 4. Auditoría y Monitoreo

- Habilitar logs de acceso
- Revisar accesos anómalos
- Alertas automáticas

**5. Uso de Roles**

- Crear roles por función de negocio
- Evitar asignación directa de privilegios

**6. Restricción de Conexiones**

- Limitar hosts de conexión
- Usar VPN para acceso remoto

**7. Encriptación**

- Encriptar datos sensibles
- Usar conexiones cifradas (SSL/TLS)

**8. Documentación**

- Documentar estructura de seguridad
- Mantener inventario de usuarios

**9. Revisiones Periódicas**

- Auditorías de seguridad regulares
- Limpieza de cuentas no utilizadas

---

# RESUMEN DEL CAPÍTULO 2

## Conceptos Clave

**DDL**: Define estructura (CREATE, ALTER, DROP, TRUNCATE) **DML**: Manipula datos (INSERT, UPDATE, DELETE) **DQL**: Consulta datos (SELECT) **Funciones**: Operaciones sobre datos **DCL**:

Controla acceso (GRANT, REVOKE)

### Orden de Ejecución SELECT

1. FROM / JOIN
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. DISTINCT
7. ORDER BY
8. LIMIT / OFFSET

### Diferencias entre SGBD

| Aspecto | Oracle | SQL Server | PostgreSQL | MySQL |
|---|---|---|---|---|
| Esquema | = Usuario | Independiente | Independiente | = Database |
| Secuencias | SEQUENCE | IDENTITY | SERIAL | AUTO_INCREMENT |
| Concat | \|\| | + | \|\| | CONCAT |
| Fecha | SYSDATE | GETDATE() | NOW() | NOW() |
| Límite | ROWNUM | TOP | LIMIT | LIMIT |

---

## GLOSARIO

**Agregación**: Combinación de valores en uno solo **Atomicidad**: Transacción completa o nada

**Cardinalidad**: Número de filas en relación **Clave Foránea**: Referencia a otra tabla **Clave Primaria**:

Identificador único de fila **CTE**: Consulta temporal nombrada **DDL**: Lenguaje de definición de datos **DML**: Lenguaje de manipulación de datos **Granularidad**: Nivel de detalle en permisos **Índice**: Estructura para acelerar búsquedas **Join**: Combina filas de tablas **NULL**: Ausencia de valor **Privilegio**: Permiso específico **Rol**: Conjunto de privilegios **Schema**: Contenedor de objetos **Subconsulta**: SELECT dentro de SELECT **Transacción**: Unidad lógica de trabajo **Vista**: Tabla virtual