Resumen Completo: Capítulo 1 - Modelos de Datos

Este documento introduce los conceptos fundamentales del modelado de datos, abarcando desde la definición básica de un sistema de base de datos hasta el proceso completo de diseño e implementación, detallando la transición de lo conceptual a lo físico.

1.1 Introducción a los Sistemas de Base de Datos (SBD)

- **Definición de SBD:** Un conjunto integrado y controlado de componentes (los propios datos, el software de gestión, los usuarios y las aplicaciones) que colaboran para permitir almacenar, organizar, gestionar de forma centralizada y recuperar datos de manera eficiente y segura.
- **Componentes Principales:**

1. **Base de Datos (BD):** La colección organizada de datos interrelacionados que representan la información de interés.
2. **SGBD (Sistema Gestor de Base de Datos):** El software crucial que actúa como intermediario entre los usuarios/aplicaciones y la base de datos física. Se encarga de todas las operaciones (Ej: PostgreSQL, Oracle, MySQL).
3. **Usuarios:** Personas o roles que interactúan con el sistema, cada uno con diferentes permisos (Ej: Estudiantes consultan, Administradores modifican).
4. **Aplicaciones:** Programas de software (Portal Web, App Móvil) que proporcionan una interfaz amigable para que los usuarios accedan y manipulen los datos.

- **Ventajas vs. Archivos Tradicionales:** Los sistemas de archivos tradicionales (ej: múltiples archivos de Excel o .txt) sufren de alta redundancia (datos duplicados), lo que inevitablemente lleva a inconsistencias (datos contradictorios), y su mantenimiento es complejo y propenso a errores. Los SBD resuelven esto al ofrecer:
  - **Reducción de redundancia:** Al centralizar los datos (ej: el nombre de un estudiante existe una sola vez), se ahorra espacio y se elimina la raíz de la inconsistencia.
  - **Consistencia e Integridad:** El SGBD impone reglas de negocio (restricciones) que aseguran la validez de los datos (ej: una nota debe estar entre 0 y 100; no se puede matricular un estudiante que no existe).
  - **Seguridad:** Se implementa un control de acceso granular, definiendo quién puede ver o modificar qué datos.
  - **Concurrencia:** El SGBD gestiona accesos simultáneos de múltiples usuarios, evitando que interfieran entre sí (ej: dos administradores intentando modificar la misma nota al mismo tiempo).
  - **Respaldo y Recuperación:** Se proporcionan mecanismos centralizados y robustos para crear copias de seguridad y restaurar los datos en caso de fallos de hardware o software.

1.2 Modelo Conceptual de Datos: Modelo Entidad-Relación (E-R)

- **Definición de E-R:** Es un modelo de datos de alto nivel (conceptual) que representa la estructura lógica de una base de datos. Utiliza entidades, atributos y relaciones para describir la información. Su principal valor es que es independiente de cualquier SGBD específico y sirve como una herramienta de comunicación fundamental entre los diseñadores y los usuarios finales (expertos del negocio).
- **Componentes del Modelo E-R:**

1. **Entidades:** Objetos o conceptos del mundo real sobre los que se desea almacenar información (Ej: `LIBRO`, `AUTOR`, `CLIENTE`).
- **Entidad Fuerte:** Tiene existencia independiente y se identifica por sus propios atributos (rectángulo simple).
- **Entidad Débil:** Depende de la existencia de otra entidad (su "dueña") para identificarse (rectángulo doble). Ej: `HABITACION` podría ser débil de `HOTEL`.
2. **Atributos:** Propiedades o características que describen a las entidades (Ej: `Título` de un `LIBRO`).
- **Simple:** Atómico, no se puede dividir (Ej: `Precio`).
- **Clave:** Un atributo (o conjunto) cuyo valor es único para cada instancia de la entidad (subrayado). Es el identificador.
- **Compuesto:** Se puede dividir en subpartes más pequeñas que tienen significado propio (Ej: `Dirección` se descompone en `Calle`, `Ciudad`, `Código Postal`).
- **Multivaluado:** Puede tener múltiples valores para una misma entidad (círculo doble). Ej: `Teléfonos` de una persona. Este tipo requerirá un tratamiento especial en el modelo lógico.
- **Derivado:** Su valor se puede calcular o inferir a partir de otro atributo (círculo punteado). Ej: `Edad` se calcula a partir de `fecha_nacimiento` y la fecha actual.
3. **Relaciones:** Asociaciones o vínculos entre dos o más entidades (representadas por un rombo). Indican cómo interactúan las entidades.
- **Cardinalidades en las Relaciones:** Definen las reglas de negocio sobre cuántas instancias de una entidad pueden (o deben) relacionarse con instancias de otra.
  - **1:1 (Uno a Uno):** Una instancia de A se relaciona con exactamente una de B, y viceversa. (Ej: `EMPLEADO` tiene asignada una `OFICINA`, y una oficina solo puede ser de un empleado).
  - **1:N (Uno a Muchos):** Una instancia de A se relaciona con cero, una o muchas de B, pero una de B solo se relaciona con una de A. (Ej: `DEPARTAMENTO` tiene muchos `EMPLEADOS`, pero un empleado pertenece a un solo departamento).
  - **N:M (Muchos a Muchos):** Una instancia de A puede relacionarse con muchas de B, y una de B puede relacionarse con muchas de A. (Ej: `ESTUDIANTE` se matricula en muchas `MATERIAS`, y una materia tiene muchos estudiantes).
- **Ejemplo de Hospital:** Se muestra un diagrama E-R completo para un hospital con entidades `PACIENTE`, `MÉDICO`, `CITA` y `TRATAMIENTO`, y sus relaciones (`solicita`, `atiende`, `prescribe`), especificando las cardinalidades que rigen su funcionamiento.

1.3 Modelo Lógico (Relacional) y Modelo Físico

Modelo Lógico (Relacional)

- **Definición:** Es el siguiente paso de abstracción. Traduce el modelo E-R a un modelo basado en tablas (llamadas **relaciones**), compuestas por filas (**tuplas**) y columnas (**atributos**). Sigue siendo independiente del SGBD específico.
- **Conceptos Clave:**
  - **Dominio:** El conjunto de valores permitidos para un atributo (ej: el dominio de "DíaSemana" es {Lunes, Martes,...}).

- **Relaciones entre tablas:** No se logran con líneas como en el E-R, sino mediante el uso de claves foráneas.
- **Restricciones del Modelo Relacional:** Son reglas cruciales para mantener la integridad de los datos.
- **Clave Primaria (PK):** Una o más columnas que identifican unívocamente cada fila (tupla) en una tabla. No puede ser nula.
- **Clave Foránea (FK):** Una columna (o conjunto) en una tabla que referencia la PK de otra tabla. Es el mecanismo que "conecta" las tablas y asegura la **integridad referencial**.
- **Integridad Referencial:** Garantiza que una fila "hija" (con FK) no pueda existir sin su fila "padre" correspondiente (con PK). Evita los "registros huérfanos" (ej: no puede existir una `MATRICULA` para un `ID_Estudiante` que no existe en la tabla `ESTUDIANTE`).
- **Integridad de Entidad:** Asegura que la Clave Primaria no puede contener valores nulos, ya que se usa para identificar la fila.

Modelo Físico de Datos

- **Definición:** Describe cómo se almacenan físicamente los datos en el hardware (discos). Este modelo es totalmente **dependiente del SGBD específico** (PostgreSQL, Oracle, etc.).
- **Jerarquía de Almacenamiento:** Los datos en el Disco Duro se organizan en **Bloques** (la unidad mínima de transferencia entre disco y memoria), que contienen **Páginas** (unidad de gestión del SGBD), las cuales almacenan los **Registros** (filas).
- **Aspectos del Modelo Físico:**
- **Tipos de datos específicos:** Se elige el tipo de dato concreto del SGBD (Ej: `SERIAL` en PostgreSQL vs. `AUTO_INCREMENT` en MySQL para autoincrementales; `VARCHAR2` en Oracle vs. `NVARCHAR` en SQL Server).
- **Estructuras de Índices:** Son estructuras de datos auxiliares que optimizan la velocidad de las búsquedas. El más común es el **Índice B-Tree**, una estructura de árbol balanceada que permite encontrar datos rápidamente (en tiempo logarítmico) sin tener que escanear la tabla completa (lo cual sería mucho más lento).
- **Particionamiento:** Técnica para dividir tablas muy grandes en piezas más pequeñas y manejables (particiones), pero que lógicamente se ven como una sola tabla. (Ej: particionamiento horizontal de `VENTAS` por año). Las consultas que filtran por año solo escanean la partición relevante, mejorando drásticamente el rendimiento.
- **Clustering:** Ordenar y almacenar físicamente los registros en el disco según un criterio de acceso frecuente (ej: agrupar todos los empleados por su `ID_Departamento`). Esto acelera mucho las consultas que buscan por ese criterio, ya que los datos están juntos.
- **Compresión de datos:** Aplicar algoritmos (ej: por diccionario) para reducir el espacio físico que ocupan los datos en disco, ahorrando costos de almacenamiento.

1.4 Mapeo del Esquema Conceptual al Relacional

- **Proceso de Transformación:** Es el flujo de diseño formal: se parte del Modelo Conceptual (E-R), se aplican reglas para convertirlo al Modelo Lógico (Tablas Relacionales), y finalmente este se implementa en el Modelo Físico (Scripts SQL específicos del SGBD).
- **Reglas de Mapeo (E-R a Relacional):**

1. **Regla 1 (Entidades Regulares):** Cada entidad se convierte en una tabla. Sus atributos simples se convierten en columnas y su clave en la Clave Primaria (PK).
2. **Regla 2 (Relaciones 1:N):** La Clave Primaria (PK) de la entidad del lado "1" se agrega como Clave Foránea (FK) a la tabla de la entidad del lado "N". Así se establece el enlace.
3. **Regla 3 (Relaciones N:M):** Se crea una nueva tabla (llamada tabla intermedia o asociativa) dedicada a la relación. Esta tabla contendrá las PK de ambas entidades originales como FK. La combinación de estas FK suele formar la PK compuesta de la nueva tabla. Es la única forma de representar el "muchos a muchos".
4. **Regla 4 (Atributos Multivaluados):** Se crea una tabla separada para el atributo. Esta tabla contendrá la PK de la entidad original (como FK) y una columna para el valor del atributo. Esto respeta la Primera Forma Normal (1NF) al no almacenar múltiples valores en una sola celda.

- **Ejemplo de Mapeo:** Se transforma un E-R de una biblioteca (`USUARIO`, `LIBRO`, `AUTOR`, `CATEGORIA` y relaciones N:M como `PRESTAMO` y `escribe`) en un esquema relacional de 6 tablas (4 por entidades, 2 por relaciones N:M).

Temas Adicionales del Capítulo

- **Niveles de Abstracción (Pirámide):** Esta separación de niveles (Conceptual, Lógico, Físico) es clave en el diseño de bases de datos. Permite manejar la complejidad (separación de preocupaciones): los usuarios se enfocan en el Conceptual, los diseñadores en el Lógico, y los administradores (DBA) en el Físico, sin que los cambios en un nivel (ej: optimizar un índice físico) afecten a los niveles superiores.

1. **Nivel Conceptual (E-R):** QUÉ datos almacenar. Orientado al usuario, independiente de tecnología.
2. **Nivel Lógico (Relacional):** CÓMO se estructuran los datos. Define tablas, PK, FK. Independiente del SGBD.
3. **Nivel Físico (Implementación):** DÓNDE se almacenan. Dependiente del SGBD (SQL, índices, tipos de datos).

- **Flujo Completo de Diseño de Base de Datos (5 Etapas):**

1. **Análisis de Requerimientos:** Entender el negocio (Entrevistas).
2. **Modelo Conceptual:** Dibujar el Diagrama E-R.
3. **Modelo Lógico:** Aplicar reglas de mapeo y **Normalización** (proceso para eliminar redundancia y evitar anomalías de actualización/borrado).
4. **Modelo Físico:** Seleccionar SGBD, definir tipos exactos, y crear índices para optimizar el rendimiento.
5. **Base de Datos Funcionando:** Implementación (correr Scripts SQL), pruebas y despliegue.

- **Herramientas Recomendadas:** Se listan herramientas de software específicas para cada nivel (Ej: Draw.io para E-R, PowerDesigner para E-R y Lógico, MySQL Workbench para Lógico y Físico, pgAdmin para Físico).
- **Criterios de Calidad:**
  - **Conceptual:** Completitud (¿Están todos los requerimientos?), consistencia. Si falla, el sistema no hará lo que el usuario necesita.
  - **Lógico:** Normalización (¿Evita anomalías?), integridad. Si falla, los datos pueden volverse corruptos o inconsistentes.

- **Físico:** Rendimiento (¿Es rápido?), escalabilidad (¿Soportará más datos?). Si falla, el sistema será lento o dejará de funcionar con el tiempo.
- **Errores Comunes:** Confundir entidades con atributos (un error común que lleva a modelos inflexibles), definir cardinalidades incorrectas (refleja mal el negocio), no normalizar (causa problemas de datos a largo plazo), usar tipos de datos inadecuados (desperdicia espacio o trunca datos), y la falta de índices en columnas de búsqueda frecuente (causa lentitud).
- **Casos de Estudio:** Se proponen 3 escenarios de práctica: Gestión Académica, Gestión Hospitalaria y Biblioteca Digital.
- **Comparación de SGBD:** Se tabulan diferencias clave entre Oracle, SQL Server, PostgreSQL, MySQL y Firebird (Licencia, sintaxis de autoincremento, tipos de datos).
- **Recursos Adicionales:** Se mencionan herramientas y la bibliografía clave (libro de Elmasri & Navathe).

Resumen Completo: Capítulo 2 - Teoría SQL

Este documento ofrece una visión integral del Lenguaje Estructurado de Consultas (SQL), cubriendo sus sublenguajes, comandos principales, funciones y conceptos de seguridad.

Introducción al SQL

SQL (Structured Query Language) es el lenguaje estándar para interactuar con bases de datos relacionales. Es un lenguaje **declarativo**, lo que significa que el usuario especifica *qué* desea obtener, no *cómo* debe la base de datos obtenerlo. El SGBD (Sistema Gestor de Base de Datos) utiliza su *optimizador* interno para determinar el plan de ejecución más eficiente (qué índices usar, cómo unir las tablas, etc.) para cumplir con la solicitud.

SQL se divide en cinco categorías principales, cada una con un propósito claro en el ciclo de vida de los datos:

1. **DDL (Data Definition Language):** Es el "plano" de la base de datos. Define la *estructura* y el esquema (tablas, columnas, tipos de datos).
   - Comandos: `CREATE`, `ALTER`, `DROP`, `TRUNCATE`.
2. **DML (Data Manipulation Language):** Se encarga del "contenido". Manipula los *datos* que viven dentro de las tablas.
   - Comandos: `INSERT`, `UPDATE`, `DELETE`.
3. **DQL (Data Query Language):** Es el lenguaje de "lectura". Realiza *consultas* para recuperar datos.
   - Comando: `SELECT`.
4. **DCL (Data Control Language):** Actúa como el "guardia de seguridad". Gestiona los *permisos* y quién puede acceder o modificar qué.
   - Comandos: `GRANT`, `REVOKE`.
5. **TCL (Transaction Control Language):** Es la "red de seguridad". Gestiona la integridad de las *transacciones* DML.
   - Comandos: `COMMIT`, `ROLLBACK`, `SAVEPOINT`.

2.1 SQL DDL (Lenguaje de Definición de Datos)

El DDL se utiliza para crear y modificar la estructura de los objetos de la base de datos. Estos comandos suelen ser *auto-commit*, lo que significa que sus cambios son permanentes e (generalmente) no se pueden deshacer con un `ROLLBACK`.

Tipos de Datos

Definen la naturaleza de la información que puede almacenar una columna. Elegir el tipo de dato correcto (ej: `SMALLINT` vs `BIGINT`) es crucial para la *integridad* (no permitir texto en un campo de edad) y la *eficiencia* (no desperdiciar espacio).

- **Numéricos:** `INTEGER` (o `INT`), `DECIMAL` (o `NUMERIC`) (para precisión exacta, como dinero), `FLOAT`, `DOUBLE`.
- **Cadenas:** `CHAR(n)` (longitud fija, rellena con espacios), `VARCHAR(n)` (longitud variable, más eficiente para textos variables), `TEXT`, `BLOB` (para datos binarios como imágenes).
- **Fecha y Hora:** `DATE`, `TIME`, `DATETIME`, `TIMESTAMP` (a menudo incluye zona horaria).
- **Especiales:** `BOOLEAN`, `JSON`, `XML`.

Restricciones de Integridad

Son reglas a nivel de tabla o columna que garantizan la validez y consistencia de los datos, previniendo la entrada de datos "basura".

- **`NOT NULL`:** La columna no puede contener valores nulos (desconocidos).
- **`DEFAULT`:** Asigna un valor predeterminado si no se especifica uno durante un `INSERT`.
- **`CHECK`:** Valida que el valor cumpla una condición lógica (ej: `precio > 0` o `stock >= 0`).
- **`PRIMARY KEY` (Clave Primaria):** Identificador único para cada fila. Es la restricción más importante; asegura que cada fila sea única e identificable.
- **`UNIQUE` (Unicidad):** Asegura que todos los valores en la columna sean únicos (similar a una PK, pero puede haber varias por tabla y generalmente permite un valor `NULL`).
- **`FOREIGN KEY` (Clave Foránea):** Es el pilar de la *integridad referencial*. Vincula una columna (la FK) con la Clave Primaria (PK) de otra tabla. Esto asegura que no se pueda, por ejemplo, insertar un `pedido` para un `cliente_id` que no existe en la tabla `clientes`.
    - **Acciones Referenciales:** Definen qué sucede si se elimina/actualiza el registro "padre" (la PK):
        - `RESTRICT` (o `NO ACTION`): (La más segura y común). Impide la operación si existen registros "hijos" (FKs) que dependen de él.
        - `CASCADE`: (Potente pero peligrosa). Propaga el cambio. Si se elimina el "padre", todos los "hijos" se eliminan automáticamente.
        - `SET NULL`: "Otorga" a los registros hijos. Si se elimina el "padre", el valor de la FK en los "hijos" se establece en `NULL`.

Comandos DDL

- **`CREATE`:** Se usa para crear nuevos objetos (`CREATE TABLE`, `CREATE INDEX`, `CREATE VIEW`).

  Ejemplo `CREATE`: Este script crea dos tablas, `Clientes` y `Pedidos`, estableciendo una Clave Primaria en `Clientes` y una Clave Foránea en `Pedidos` que referencia a `Clientes`.

```
-- Crea la tabla de Clientes
CREATE TABLE Clientes (
    cliente_id INT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    fecha_registro DATE DEFAULT CURRENT_DATE
);

-- Crea la tabla de Pedidos con una FK a Clientes
CREATE TABLE Pedidos (
    pedido_id INT PRIMARY KEY,
    cliente_id INT,
    monto DECIMAL(10, 2),
    fecha_pedido DATE,

    CONSTRAINT fk_cliente
        FOREIGN KEY (cliente_id)
        REFERENCES Clientes(cliente_id)
        ON DELETE SET NULL
);
```

- **`ALTER TABLE`:** Modifica una tabla existente (`ADD COLUMN`, `DROP COLUMN`, `MODIFY COLUMN`, `ADD CONSTRAINT`). Es una operación que puede ser lenta en tablas muy grandes.

  Ejemplo `ALTER`: Este script añade una nueva columna `telefono` a la tabla `Clientes` que creamos.

```
-- Añade una nueva columna a la tabla Clientes
ALTER TABLE Clientes
ADD COLUMN telefono VARCHAR(20);
```

- **`DROP`:** Elimina permanentemente un objeto (`DROP TABLE`, `DROP INDEX`). Esta acción es definitiva.

Ejemplo `DROP`: Este script elimina permanentemente la tabla `Pedidos`. Si `Clientes` tuviera una FK apuntando a `Pedidos`, esta operación fallaría a menos que se use `CASCADE`.

```
-- Elimina la tabla Pedidos
DROP TABLE Pedidos;
```

- **`TRUNCATE TABLE`:** Elimina *todos* los datos de una tabla. Es mucho más rápido que `DELETE` porque no registra cada fila borrada individualmente (no activa triggers); simplemente desasigna las páginas de datos.

  Ejemplo `TRUNCATE`: Este script vacía la tabla `Clientes` instantáneamente, pero mantiene su estructura (columnas, índices, etc.) intacta.

```
-- Vacía todos los registros de la tabla Clientes
TRUNCATE TABLE Clientes;
```

Índices

Estructuras (comúnmente **B-Tree**) que mejoran drásticamente la velocidad de las consultas (`SELECT`, `JOIN`, `WHERE`). Funcionan como el índice de un libro: permiten al SGBD "saltar" directamente al dato solicitado sin escanear la tabla entera.

- **El Costo (Trade-off):** Los índices ralentizan las operaciones de escritura (`INSERT`, `UPDATE`, `DELETE`), ya que cada vez que se modifica un dato, el SGBD debe actualizar tanto la tabla como el índice.

  Ejemplo `CREATE INDEX`: Si buscamos clientes frecuentemente por `email`, este script crea un índice para acelerar esas búsquedas.

```
-- Crea un índice en la columna email para búsquedas rápidas
CREATE INDEX idx_email_cliente
ON Clientes (email);
```

2.2 SQL DML (Lenguaje de Manipulación de Datos)

El DML se utiliza para insertar, modificar y eliminar los datos *dentro* de las tablas. Estas operaciones son transaccionales y se pueden deshacer.

- **`INSERT INTO tabla (col1, col2) VALUES (val1, val2);`**
  - Añade nuevas filas. Puede insertar múltiples filas o el resultado de un `SELECT`.

  Ejemplo `INSERT`: Este script añade un nuevo cliente a nuestra tabla.

```
-- Inserta un nuevo registro en Clientes
INSERT INTO Clientes (cliente_id, nombre, email, telefono)
VALUES (1, 'Juan Pérez', 'juan.perez@email.com', '555-1234');
```

- **`UPDATE tabla SET col1 = val1 WHERE condición;`**
  - Modifica datos existentes. **¡El `WHERE` es crucial!** La consecuencia de omitir el `WHERE` es actualizar *todas las filas* de la tabla, un error catastrófico común.

    Ejemplo `UPDATE`: Este script actualiza el teléfono del cliente que acabamos de insertar.
    ```
    -- Actualiza el teléfono de un cliente específico
    UPDATE Clientes
    SET telefono = '555-5678'
    WHERE cliente_id = 1;
    ```

- **`DELETE FROM tabla WHERE condición;`**
  - Elimina filas. Al igual que con `UPDATE`, omitir el `WHERE` elimina *todas las filas*. `DELETE` es más lento que `TRUNCATE` porque es una operación registrada fila por fila y activa triggers.

    Ejemplo `DELETE`: Este script elimina al cliente con `cliente_id = 1`.
    ```
    -- Elimina un cliente específico
    DELETE FROM Clientes
    WHERE cliente_id = 1;
    ```

    Transacciones (TCL)

    El DML está indisolublemente ligado al TCL. Una transacción (iniciada implícita o explícitamente) es una unidad de trabajo que debe ser **atómica**: o se completa en su totalidad, o no se completa en absoluto ("todo o nada").

- **Ejemplo clásico:** Una transferencia bancaria. Son dos `UPDATE` (restar dinero de la cuenta A, sumar dinero a la cuenta B). Si el primer `UPDATE` funciona pero el segundo falla, la transacción entera debe deshacerse.
- `COMMIT`: Confirma y guarda permanentemente todos los cambios de la transacción.
- `ROLLBACK`: Deshace todos los cambios realizados desde el último `COMMIT`.

  Ejemplo `TCL`: Simula una transferencia bancaria. Si algo falla antes del `COMMIT`, el `ROLLBACK` desharía ambos `UPDATE`, asegurando que el dinero no se "pierda".
  ```
  START TRANSACTION;

  -- Restar 100 de la cuenta 1
  UPDATE Cuentas SET saldo = saldo - 100 WHERE cuenta_id = 1;

  -- Sumar 100 a la cuenta 2
  UPDATE Cuentas SET saldo = saldo + 100 WHERE cuenta_id = 2;
  ```

```
-- Si todo salió bien, confirmar los cambios
COMMIT;
-- Si algo hubiera fallado, se ejecutaría:
-- ROLLBACK;
```

2.3 SQL DQL (Lenguaje de Consulta de Datos)

El comando `SELECT` es el núcleo de DQL, usado para recuperar datos.

Estructura y Orden de Ejecución Lógico

El orden en que se *escribe* la consulta es diferente al orden en que la base de datos la *procesa*. Entender esto es clave para evitar errores.

1. **`FROM` / `JOIN`:** Identifica las tablas y las combina.
2. **`WHERE`:** Filtra las filas individuales.
3. **`GROUP BY`:** Agrupa las filas con valores idénticos.
4. **`HAVING`:** Filtra los *grupos* ya formados.
5. **`SELECT`:** Elige las columnas, calcula expresiones y asigna alias.
6. **`DISTINCT`:** Elimina filas duplicadas del resultado final.
7. **`ORDER BY`:** Ordena el resultado final (`ASC` o `DESC`).
8. **`LIMIT` / `OFFSET`:** Limita el número de filas devueltas.

- **Implicación clave:** No se puede usar un alias definido en `SELECT` (Paso 5) dentro de un `WHERE` (Paso 2), porque `WHERE` se procesa *antes*.

Ejemplo Básico `SELECT`: Este script recupera el nombre y email de todos los clientes registrados este año, los ordena por nombre y solo muestra los primeros 10.

```
SELECT nombre, email
FROM Clientes
WHERE EXTRACT(YEAR FROM fecha_registro) = 2024
ORDER BY nombre ASC
LIMIT 10;
```

Operaciones `JOIN`

Combinan filas de dos o más tablas:

- **`INNER JOIN`:** El más común. Devuelve solo las filas que tienen una coincidencia exacta en ambas tablas.
- **`LEFT JOIN`:** Devuelve *todas* las filas de la tabla izquierda, y las coincidencias de la derecha (o `NULL` si no hay). Esencial para preguntas como "mostrar todos los clientes y *cualquier* pedido que tengan (incluso si no tienen ninguno)".

- **`RIGHT JOIN`:** Inverso del `LEFT JOIN`. Devuelve todas las filas de la tabla derecha.
- **`FULL OUTER JOIN`:** Devuelve *todas* las filas de ambas tablas, rellenando con `NULL` donde no hay coincidencia. Se usa en análisis y data warehousing.
- **`SELF JOIN`:** Una tabla se une consigo misma (usando alias) para encontrar relaciones jerárquicas (ej: "mostrar cada empleado y el nombre de su jefe", donde el jefe está en la misma tabla).

  Ejemplo `JOIN`: Este script usa `INNER JOIN` para mostrar los nombres de los clientes junto con los montos de los pedidos que han realizado.
  ```
  SELECT c.nombre, p.monto
  FROM Clientes c
  INNER JOIN Pedidos p ON c.cliente_id = p.cliente_id;
  ```

  Ejemplo `LEFT JOIN`: Este script mostraría *todos* los clientes, tengan o no pedidos. Si un cliente no tiene pedidos, el `monto` aparecerá como `NULL`.
  ```
  SELECT c.nombre, p.monto
  FROM Clientes c
  LEFT JOIN Pedidos p ON c.cliente_id = p.cliente_id;
  ```

  Agregación y Agrupamiento

- **Funciones de Agregación:** Operan sobre un conjunto de filas y devuelven un solo valor: `COUNT()` (contar), `SUM()` (sumar), `AVG()` (promedio), `MIN()` (mínimo), `MAX()` (máximo).
- **`GROUP BY`:** Se usa junto con las funciones de agregación para "colapsar" filas en grupos (ej: `SUM(ventas) GROUP BY region` da el total por región).
- **`WHERE` vs. `HAVING`:** `WHERE` filtra *filas individuales* (ej: `WHERE anio = 2024`) *antes* de agrupar. `HAVING` filtra *grupos* (ej: `HAVING SUM(ventas) > 10000`) *después* de agrupar.

  Ejemplo `GROUP BY` y `HAVING`: Este script cuenta cuántos pedidos ha hecho cada cliente y, al final, solo muestra a los clientes que han hecho más de 5 pedidos.
  ```
  SELECT
      cliente_id,
      COUNT(pedido_id) AS total_pedidos
  FROM Pedidos
  GROUP BY cliente_id
  HAVING COUNT(pedido_id) > 5;
  ```

  Subconsultas (Subqueries)

  Una consulta `SELECT` anidada dentro de otra. Pueden usarse en:

- `WHERE`: Para filtrar (ej: `WHERE id IN (SELECT ...)`).
- `FROM`: Como una tabla temporal (ej: `FROM (SELECT ...) AS t`).
- `SELECT`: Como una columna calculada (debe devolver un solo valor).
- Operadores comunes: `IN`, `EXISTS`, `ANY`, `ALL`.

Ejemplo `Subconsulta`: Este script obtiene los nombres de todos los clientes que han realizado al menos un pedido, usando una subconsulta en el `WHERE`.

```
SELECT nombre
FROM Clientes
WHERE cliente_id IN (SELECT DISTINCT cliente_id FROM
Pedidos);
```

Operadores de Conjunto

Combinan los resultados de dos consultas (que deben tener el mismo número y tipo de columnas):

- **`UNION`:** Combina resultados y realiza un trabajo extra para eliminar duplicados.
- **`UNION ALL`:** Simplemente apila los resultados, incluyendo duplicados. Es mucho más rápido que `UNION` si se sabe que no hay duplicados o si no importan.
- **`INTERSECT`:** Devuelve solo las filas que están presentes en *ambos* resultados.
- **`EXCEPT` / `MINUS`:** Devuelve las filas de la primera consulta que *no* están en la segunda.

Ejemplo `UNION ALL`: Este script crea una lista única de contactos combinando clientes y proveedores (asumiendo que ambas tablas tienen columnas `nombre` y `email`).

```
SELECT nombre, email FROM Clientes
UNION ALL
SELECT nombre_contacto, email FROM Proveedores;
```

Funciones Avanzadas de DQL

- **CTEs (Common Table Expressions):** `WITH nombre_cte AS (SELECT ...)`: Crea una consulta temporal nombrada. Son excelentes para organizar consultas largas y complejas, haciéndolas más legibles. También son la base para consultas recursivas (ej: organigramas).

Ejemplo `CTE`: Este script usa un CTE (`PedidosCliente`) para pre-calcular el total de pedidos por cliente, y luego une ese resultado a la tabla de Clientes.

```
WITH PedidosCliente AS (
    SELECT cliente_id, COUNT(pedido_id) AS total_pedidos
    FROM Pedidos
    GROUP BY cliente_id
)
SELECT c.nombre, pc.total_pedidos
```

```
FROM Clientes c
JOIN PedidosCliente pc ON c.cliente_id = pc.cliente_id;
```

- **Funciones de Ventana:** Usan la cláusula `OVER()`. Realizan cálculos sobre un "marco" o "ventana" de filas (ej: `PARTITION BY`, `ORDER BY`). Su gran ventaja es que **no colapsan el resultado** como `GROUP BY`. Mantienen todas las filas originales y añaden columnas de cálculo (ej: un ranking de salarios por departamento, una suma acumulada de ventas por mes).
  - **Ranking:** `ROW_NUMBER()`, `RANK()`, `DENSE_RANK()`.
  - **Valor:** `LAG()` (valor de la fila anterior), `LEAD()` (valor de la fila siguiente).

  Ejemplo `Función de Ventana`: Este script muestra el salario de cada empleado, junto con el salario *promedio de su departamento*, sin colapsar las filas.

```
SELECT
    nombre,
    departamento,
    salario,
    AVG(salario) OVER (PARTITION BY departamento) AS
promedio_depto
FROM Empleados;
```

2.4 Funciones del SGBD

Operaciones predefinidas para transformar datos dentro de un `SELECT` o `WHERE`.

- **Condicionales:**
  - `CASE WHEN condición THEN res1 ELSE res2 END`: Lógica `if/then/else` versátil dentro de un `SELECT`.
- **Funciones de Cadenas:**
  - `UPPER()`, `LOWER()`, `TRIM()` (quitar espacios), `SUBSTRING()`, `LENGTH()`, `REPLACE()`, `CONCAT()`.
- **Funciones Matemáticas:**
  - `ROUND()`, `CEILING()` (redondear hacia arriba), `FLOOR()` (redondear hacia abajo), `ABS()` (valor absoluto), `MOD()` (módulo/resto), `POWER()` (potencia).
- **Funciones de Fecha y Hora:**
  - `NOW()`, `GETDATE()`, `CURRENT_TIMESTAMP` (obtener fecha/hora actual).
  - `EXTRACT()` (extraer partes, ej: `YEAR`), `DATE_ADD()`, `DATEDIFF()`, `TO_CHAR()` (para formateo).
- **Funciones de Conversión:**
  - `CAST(valor AS tipo)` y `CONVERT()`: Cambian explícitamente el tipo de dato (ej: de texto a número).
- **Manejo de Nulos:**
  - `NULL` significa "valor desconocido", no es cero ni una cadena vacía.

- `COALESCE(val1, val2, ...)`: Esencial para la limpieza de datos. Devuelve el primer valor no nulo de la lista. Se usa comúnmente para reemplazar `NULL` con un valor predeterminado (ej: `COALESCE(descuento, 0)`).
- `ISNULL()`, `IFNULL()`, `NULLIF()` (devuelve `NULL` si dos valores son iguales).

Ejemplo de Funciones Múltiples: Este script demuestra varias funciones en una sola consulta.

```
SELECT
    UPPER(nombre) AS nombre_mayus,
    COALESCE(telefono, 'No disponible') AS telefono_limpio,
    ROUND(monto, 0) AS monto_redondeado,
    CASE
        WHEN monto > 1000 THEN 'Pedido Grande'
        WHEN monto > 500 THEN 'Pedido Mediano'
        ELSE 'Pedido Pequeño'
    END AS categoria_pedido
FROM Pedidos;
```

2.5 SQL DCL (Lenguaje de Control de Datos)

El DCL gestiona la seguridad y los permisos de acceso, asegurando que los usuarios solo puedan ver y hacer lo que tienen permitido.

- **Usuarios (`USER`):** Identidades que se conectan a la BD. Se crean con `CREATE USER`.
- **Roles (`ROLE`):** Un conjunto de privilegios. Los roles son la mejor práctica para la gestión de permisos. En lugar de dar 50 permisos a 100 usuarios (5000 `GRANT`s), se crea un rol (ej: `rol_desarrollador`), se le dan los 50 permisos, y luego se asigna ese único rol a los 100 usuarios. Es inmensamente más fácil de mantener.
- **Esquemas (`SCHEMA`):** Un contenedor lógico (como una carpeta) que agrupa objetos (tablas, vistas, etc.). Ayuda a organizar la base de datos y a aplicar permisos a nivel de contenedor.

Comandos DCL

- **`GRANT`:** Otorga permisos (privilegios) a un usuario o rol.
  - `GRANT SELECT, INSERT ON mi_tabla TO mi_usuario;` (Permiso de objeto)
  - `GRANT mi_rol TO mi_usuario;` (Asignación de rol)
  - `WITH GRANT OPTION`: Permite al usuario receptor otorgar ese mismo permiso a otros (generalmente no recomendado).

Ejemplo `GRANT`: Este script crea un rol de "solo lectura" y se lo asigna a un usuario.

```
-- Crear un rol
CREATE ROLE rol_lectura;
```

```
-- Dar permiso de LECTURA (SELECT) al rol sobre la tabla
Clientes
GRANT SELECT ON Clientes TO rol_lectura;

-- Asignar el rol a un usuario
GRANT rol_lectura TO 'usuario_reportes'@'localhost';
```

- **`REVOKE`:** Revoca permisos.
  - `REVOKE UPDATE ON mi_tabla FROM mi_usuario;`
  - `CASCADE`: Revoca el permiso y también lo revoca de cualquier otro usuario al que `mi_usuario` se lo haya otorgado (si tenía `WITH GRANT OPTION`).

Ejemplo `REVOKE`: Este script quita el permiso `INSERT` que el `rol_lectura` pudiera tener (aunque en el ejemplo anterior no se lo dimos).

```
-- Quita el permiso de INSERTAR del rol
REVOKE INSERT ON Clientes FROM rol_lectura;
```

Buenas Prácticas de Seguridad

- **Principio de Mínimo Privilegio:** Es la regla de oro. Otorgar solo los permisos estrictamente necesarios para que un usuario o aplicación realice su trabajo, y nada más. Un usuario de reportes solo necesita `SELECT`; nunca debe tener `DELETE` o `DROP`, limitando así el daño accidental o malicioso.
- Usar Roles para gestionar permisos en lugar de asignarlos directamente a usuarios.
- Gestionar contraseñas de forma segura y rotarlas periódicamente.

Resumen del Capítulo y Diferencias entre SGBD

El capítulo concluye resaltando que, aunque SQL es un estándar ANSI/ISO, en la práctica existen **dialectos** específicos para cada SGBD (Oracle, SQL Server, PostgreSQL, MySQL). Estas diferencias obligan a los desarrolladores a ser cuidadosos.

- **Ejemplos de Dialectos:**
  - **Secuencias/Autoincrementales:** `SEQUENCE` (Oracle), `IDENTITY` (SQL Server), `SERIAL` (PostgreSQL), `AUTO_INCREMENT` (MySQL).
  - **Concatenación de cadenas:** `||` (Oracle/PostgreSQL), `+` (SQL Server), `CONCAT()` (MySQL).
  - **Funciones de fecha:** `SYSDATE` (Oracle), `GETDATE()` (SQL Server), `NOW()` (PostgreSQL/MySQL).
  - **Limitación de resultados:** `ROWNUM` (Oracle), `TOP` (SQL Server), `LIMIT` (PostgreSQL/MySQL).
- **Implicación:** Estas diferencias son la razón principal por la que existen herramientas de abstracción como los ORMs (Object-Relational Mappers), que permiten escribir código en un lenguaje (como Python o Java) y dejan que el ORM genere el dialecto SQL correcto para la base de datos subyacente.