Resumen de Estudio: Dependencias Funcionales (Tema 3.1)

Este resumen está diseñado para preparar el examen teórico-práctico, cubriendo definiciones, axiomas y razonamiento lógico sobre el comportamiento de los datos.

1. Conceptos Fundamentales (Teoría Pura)

¿Qué es una Dependencia Funcional (DF)?

Es una restricción entre dos conjuntos de atributos en una relación (tabla).

- **Notación:** $X \rightarrow Y$ (X determina a Y).
- **Definición Formal:** Para cualquier par de tuplas t_1 y t_2 en una instancia $r(R)$, si $t_1[X] = t_2[X]$, entonces obligatoriamente $t_1[Y] = t_2[Y]$.
- **Analogía:** DNI $\rightarrow$ Nombre. Si dos personas tienen el mismo DNI, *tienen* que llamarse igual.

Tipos de Dependencias

Es crucial distinguir estos tipos para las preguntas teóricas y de normalización:

| **Tipo** | **Definición** | **Ejemplo** | **"Qué pasa si..." (Impacto)** | | **Trivial** | $Y \subseteq X$ (Y es parte de X). | `{DNI, Nombre} -> Nombre` | **Siempre es verdadera**. No aporta información útil para restringir datos. | | **No Trivial** | $Y \not\subseteq X$. | `DNI -> Nombre` | Aporta restricciones reales e información semántica. | | **Completa** | Y depende de **todo** X, no de una parte. | `{Estudiante, Curso} -> Nota` | Es lo deseable para las Claves Primarias. Define la **2NF**. | | **Parcial** | Y depende solo de una **parte** de X. | `{Estudiante, Curso} -> NombreEst` | **Viola la 2NF**. Causa redundancia (el nombre se repite por cada curso). | | **Transitiva** | $X \rightarrow Y \rightarrow Z$ (y Y no determina X). | `Empleado -> Depto -> Jefe` | **Viola la 3NF**. Causa anomalías (si borro al empleado, pierdo al jefe). |

2. Axiomas de Armstrong (Las Reglas del Juego)

Son reglas para inferir nuevas dependencias a partir de las existentes. Son **completas** (hallan todas las posibles) y **correctas** (no hallan falsas).

Axiomas Básicos (Primarios)

1. **Reflexividad:** Si $Y \subseteq X$, entonces $X \rightarrow Y$. (Crea dependencias triviales).
2. **Aumento:** Si $X \rightarrow Y$, entonces $XZ \rightarrow YZ$. (Si agrego lo mismo a ambos lados, se mantiene).
3. **Transitividad:** Si $X \rightarrow Y$ y $Y \rightarrow Z$, entonces $X \rightarrow Z$.

Reglas Derivadas (Útiles para ejercicios prácticos)

1. **Unión:** Si $X \rightarrow Y$ y $X \rightarrow Z$, entonces $X \rightarrow YZ$.
2. **Descomposición:** Si $X \rightarrow YZ$, entonces $X \rightarrow Y$ y $X \rightarrow Z$. (Ojo: **No** se puede descomponer el lado izquierdo).
3. **Pseudotransitividad:** Si $X \rightarrow Y$ y $WY \rightarrow Z$, entonces $WX \rightarrow Z$.

3. Algoritmos y Procedimientos (Para la parte práctica)

Cierre de un Conjunto de Atributos (X^+)

Es el conjunto de todos los atributos que pueden ser determinados funcionalmente por X.

- **¿Para qué sirve?**
  1. Para saber si una dependencia $X \to Y$ es válida (verificamos si $Y \subseteq X^+$).
  2. Para encontrar **Claves Candidatas** (si X^+ contiene *todos* los atributos de la tabla, X es superclave).

Cobertura Canónica (Conjunto Minimal)

Es la versión simplificada y equivalente de un conjunto de dependencias F. Pasos:

1. **Descomponer la derecha:** Que solo haya un atributo a la derecha de cada flecha ($A \to BC$ pasa a $A \to B$ y $A \to C$).
2. **Eliminar atributos extraños a la izquierda:** Si $AB \to C$ y resulta que $A \to C$ por sí solo, B sobra.
3. **Eliminar dependencias redundantes:** Si una regla se puede deducir usando transitividad con las otras, se borra.

4. Preguntas Tipo "Qué pasa si..." (Razonamiento)

Para el examen, prepárate para responder situaciones hipotéticas basadas en la teoría:

P: ¿Qué pasa si encuentro dos filas con el mismo valor en X pero diferente valor en Y?

- **R:** La dependencia $X \to Y$ **NO se cumple** (se viola). La instancia de la base de datos es inconsistente con esa regla.

P: ¿Qué pasa si tengo una dependencia parcial ($X \to Y$ donde X es clave compuesta)?

- **R:** Se produce **redundancia**. Los datos de Y se repetirán innecesariamente cada vez que aparezca la parte de la clave X. Esto lleva a anomalías de actualización.

P: ¿Qué pasa si elimino una tupla en una tabla con dependencias transitivas? (Anomalía de borrado)

- **R:** Podrías perder información no deseada. *Ejemplo:* Si `Empleado -> Departamento -> Ubicación`, y borras al último empleado de "Ventas", pierdes el dato de dónde está ubicado "Ventas".

P: ¿Si $A \to B$, implica esto que $B \to A$?

- **R:** No necesariamente. Solo si la relación es biyectiva (1 a 1). En general, la dependencia funcional no es conmutativa.

P: ¿Por qué la dependencia `{DNI, Nombre} -> DNI` no es útil?

- **R:** Porque es una **Dependencia Trivial**. Cualquier conjunto determina a un subconjunto de sí mismo. No restringe los datos de ninguna manera.

P: ¿Cómo demuestro que una descomposición de tablas es correcta?

- **R:** Debes verificar que la descomposición preserve la información (sin pérdida de reunión) y, idealmente, que preserve las dependencias funcionales (que no se pierdan reglas al separar atributos).

Resumen de Estudio: Claves, Superclaves y Cierre (Tema 3.2)

Este resumen organiza los conceptos jerárquicos de las claves y profundiza en la lógica necesaria para responder preguntas tipo "¿Qué pasa si...?" y ejercicios de verificación.

1. Jerarquía de Claves (Definiciones Teóricas)

Es vital entender que todos estos conceptos están contenidos unos dentro de otros.

A. Superclave (La base general)

- **Definición:** Conjunto de uno o más atributos que identifican de manera **única** a una tupla.
- **Propiedad clave: Unicidad**. (No le importa si sobran datos).
- **Regla de oro:** Si K es Superclave, cualquier conjunto que contenga a K ($K \cup Z$) también es Superclave.
- **Ejemplo:** Si `ID` es único, entonces `{ID}`, `{ID, Nombre}`, `{ID, ColorFavorito}` son todas superclaves.

B. Clave Candidata (La versión optimizada)

- **Definición:** Es una Superclave **minimal**.
- **Propiedades clave:**
  1. **Unicidad:** Identifica filas únicas.
  2. **Minimalidad:** Si quitas *cualquier* atributo del conjunto, deja de ser único.
- **Diferencia Crucial:** Una Clave Candidata **NO** tolera atributos redundantes.

C. Clave Primaria (La elegida)

- **Definición:** Es *una* de las Claves Candidatas elegida por el diseñador para ser el identificador principal.
- **Requisitos Extra:** No puede ser `NULL` (Integridad de Entidad) y preferiblemente debe ser inmutable.

D. Claves Alternativas (Las suplentes)

- **Definición:** Son las Claves Candidatas que **no** fueron elegidas como Primaria.
- **Uso:** Se implementan con restricciones `UNIQUE`.

2. Herramientas Prácticas: Cierre de Atributos (X^+)

El "Cierre" es la herramienta matemática para demostrar si algo es clave o no.

- **Definición:** X^+ es el conjunto de todos los atributos que se pueden inferir funcionalmente a partir de X usando las dependencias funcionales F.
- **Algoritmo Mental Rápido:**
  1. Empieza con el atributo que estás probando: $Resultado = \{A\}$.
  2. Busca flechas que salgan de A ($A \rightarrow B$). Añade B al resultado.

3. Ahora tienes $\{A, B\}$. Busca flechas que salgan de A, de B, o de la combinación AB.
4. Repite hasta que no puedas añadir nada más.

¿Cómo usarlo en el examen?

- **Para probar Superclave:** Calcula X^+. Si el resultado contiene **todos** los atributos de la tabla (R), entonces X es Superclave.
- **Para probar Clave Candidata:**
  1. Verifica que es Superclave.
  2. Intenta quitarle atributos uno por uno. Si al quitarle un atributo, el cierre *ya no* cubre toda la tabla, entonces ese atributo era necesario. Si ninguno se puede quitar, es Clave Candidata.

3. Razonamiento y Preguntas "Qué pasa si..."

Esta sección ataca directamente las preguntas capciosas o de análisis.

Escenario 1: Redundancia

P: ¿Qué pasa si agrego un atributo extra a una Clave Candidata?

- **R:** Se convierte en una **Superclave**, pero deja de ser Clave Candidata porque pierde la propiedad de **minimalidad**. Sigue sirviendo para identificar filas, pero no es eficiente ni correcto para el diseño lógico.

Escenario 2: Claves Compuestas

P: ¿Si `{A, B}` es clave candidata, implica que `A` y `B` son únicos por separado?

- **R: No.** Solo la combinación es única.
  - *Ejemplo:* `{Estudiante, Curso}`. Un estudiante se repite (toma muchos cursos) y un curso se repite (tiene muchos estudiantes). Solo el par es único.
  - **Ojo:** Si `A` fuera único por sí solo, entonces `B` sobraría y `{A, B}` no sería minimal (sería superclave, pero no candidata).

Escenario 3: Modificación de Valor

P: ¿Qué pasa si cambio el valor de una Clave Primaria que es referenciada por otra tabla?

- **R:** Se viola la **Integridad Referencial**. La base de datos debe rechazar el cambio o realizar una actualización en cascada (dependiendo de la configuración), o las filas hijas quedarían "huérfanas".

Escenario 4: Nulos

P: ¿Puede una parte de una Clave Primaria compuesta ser NULL?

- **R: No.** Ninguna parte de la Clave Primaria puede ser nula. Sin embargo, una Clave Alternativa (Unique) sí suele permitir nulos (dependiendo del motor de BD).

Escenario 5: Elección de Clave

P: Tengo dos claves candidatas: `DNI` (Natural) e `ID_Interno` (Artificial). ¿Cuál elijo y qué pasa con la otra?

- **R:** Generalmente se elige `ID_Interno` por **estabilidad** (no cambia) y **eficiencia** (suele ser entero). El `DNI` pasa a ser **Clave Alternativa**.
    - *Riesgo:* Si eliges DNI como primaria y el gobierno cambia el formato de los DNI o descubres un error de digitación, cambiarlo es costoso porque afecta a todas las claves foráneas que apunten a él.

4. Criterios de Selección (Checklist)

Para preguntas tipo "¿Cuál es la mejor clave primaria?":

1. **Unicidad:** (Obligatorio).
2. **Irreductibilidad:** (Obligatorio - debe ser candidata).
3. **Estabilidad:** Que no cambie con el tiempo (e.g., el email es malo porque la gente lo cambia).
4. **Simplicidad:** Preferible un solo atributo (INT) que tres (VARCHAR).
5. **Disponibilidad:** Que siempre tenga valor al momento de crear el registro (no puede ser nulo).

Resumen de Estudio: Normalización (Tema 3.3)

Este resumen está estructurado para entender la evolución de las reglas (de 1NF a BCNF) y para resolver problemas de razonamiento sobre anomalías y dependencias.

1. El "Por Qué" (Teoría de Anomalías)

Antes de memorizar reglas, entiende qué intentamos evitar. Si te preguntan "¿Por qué normalizamos?", la respuesta es: **Para minimizar redundancia y evitar anomalías**.

Las Tres Anomalías Clásicas

Imagina una tabla `EMPLEADO_DEPTO` donde guardamos datos del empleado y datos repetidos de su departamento en la misma fila.

1. **Anomalía de Inserción:**
   - *Definición:* No puedes añadir un dato X sin añadir obligatoriamente un dato Y no relacionado.
   - *Ejemplo:* No puedo crear un nuevo Departamento si aún no he contratado a ningún empleado para él (porque la clave primaria es el ID del empleado).
2. **Anomalía de Actualización:**
   - *Definición:* Datos redundantes que obligan a actualizar múltiples filas para cambiar un solo hecho.
   - *Ejemplo:* Si el departamento "Ventas" se muda de piso, tengo que actualizar esa dirección en las 1,000 filas de los empleados de ventas. Si fallo en una, los datos son inconsistentes.
3. **Anomalía de Eliminación:**
   - *Definición:* Al borrar un dato, pierdes accidentalmente otra información que querías conservar.
   - *Ejemplo:* Si despido al único empleado del departamento de "Innovación", borro la fila y accidentalmente desaparece el registro de que existe un departamento llamado "Innovación".

2. Las Formas Normales (Reglas y Violaciones)

Piensa en las formas normales como filtros progresivos. Una tabla debe cumplir la anterior para pasar a la siguiente.

Primera Forma Normal (1NF) - "Atomicidad"

- **Regla:** Sin grupos repetitivos ni valores múltiples en una celda.
- **Se viola si:** Una celda contiene "rojo, verde, azul" o hay columnas `Tel1`, `Tel2`.
- **Solución:** Crear filas separadas para cada valor o mover los repetidos a una nueva tabla.

Segunda Forma Normal (2NF) - "Todo sobre la clave completa"

- **Requisito previo:** Estar en 1NF.
- **Regla:** Eliminar **Dependencias Parciales**. Todo atributo no-clave debe depender de la **Clave Primaria COMPLETA**, no solo de una parte.

- **Cuándo preocuparse: SOLO** si tienes una **Clave Primaria Compuesta** (formada por 2+ columnas).
  - *Si la clave es simple (una sola columna), automáticamente estás en 2NF.*
- **Ejemplo de Violación:** En `INSCRIPCION (ID_Estudiante, ID_Curso, Nombre_Estudiante)`, el `Nombre_Estudiante` depende solo de `ID_Estudiante`, no del curso.
- **Solución:** Mover `Nombre_Estudiante` a una tabla de Estudiantes.

Tercera Forma Normal (3NF) - "Nada de intermediarios"

- **Requisito previo:** Estar en 2NF.
- **Regla:** Eliminar **Dependencias Transitivas**. Ningún atributo no-clave debe depender de otro atributo no-clave.
- **Fórmula del crimen:** $A \rightarrow B \rightarrow C$ (donde A es PK). El atributo C depende de B, que no es clave.
- **Ejemplo de Violación:** `EMPLEADO (ID, Nombre, Codigo_Postal, Ciudad)`. La `Ciudad` depende del `Codigo_Postal`, no directamente del empleado.
- **Solución:** Mover `Codigo_Postal` y `Ciudad` a una tabla de Direcciones.

Forma Normal de Boyce-Codd (BCNF) - "El jefe estricto"

- **Requisito previo:** Estar en 3NF.
- **Regla:** Todo determinante debe ser una **Superclave**.
- **Diferencia con 3NF:** La 3NF es permisiva si el atributo dependiente es parte de una clave candidata. BCNF no permite excepciones.
- **Cuándo ocurre la violación:** Es raro, suele pasar si:
  1. Hay claves candidatas compuestas que se solapan.
  2. Un atributo de la clave compuesta depende de un atributo que no es clave.

3. Preguntas de Razonamiento ("Qué pasa si...")

Estas son las preguntas trampa típicas del examen.

P: ¿Qué pasa si tengo una tabla con clave primaria simple (un solo campo) y me piden verificar la 2NF?

- **R:** No pierdas tiempo buscando dependencias parciales. Por definición, **ya cumple la 2NF**. Pasa directamente a verificar la 3NF.

P: ¿Qué pasa si desnormalizo una base de datos (vuelvo atrás de 3NF a 2NF o 1NF)?

- **R:**
  - *Ventaja:* Ganas velocidad en lectura (consultas `SELECT` más rápidas) porque evitas hacer `JOINs` entre muchas tablas.
  - *Desventaja:* Introduces redundancia y riesgo de anomalías de actualización. Aumentas el espacio en disco.

P: ¿Qué pasa si encuentro una dependencia transitiva $A \rightarrow B \rightarrow C$ pero no la arreglo?

- **R:** Estás violando la **3NF**. Tendrás datos repetidos de C cada vez que aparezca B. (Ej: Repetir el nombre del departamento por cada empleado).

P: ¿Puede una tabla estar en BCNF pero no en 3NF?

- **R: Imposible**. Es una jerarquía estricta. Para ser BCNF, *tienes* que ser 3NF primero.

P: ¿Qué pasa si tengo una lista de teléfonos en una sola celda separados por comas?

- **R:** Violas la **1NF** (atomicidad). Las consultas SQL serán muy ineficientes (tendrás que usar funciones de texto lentas para buscar un número).

4. Algoritmo Rápido para el Examen

Si te dan una tabla y te dicen "¿En qué forma normal está?", sigue este diagrama de flujo mental:

1. **¿Hay celdas con múltiples valores?**
   - SÍ $\rightarrow$ **No Normalizada**.
   - NO $\rightarrow$ Pasa al paso 2 (Ya es **1NF**).
2. **¿La Clave Primaria es compuesta?**
   - NO $\rightarrow$ Pasa al paso 3 (Ya es **2NF**).
   - SÍ $\rightarrow$ **¿Algún atributo depende solo de una parte de la clave?**
     - SÍ $\rightarrow$ Se queda en **1NF**.
     - NO $\rightarrow$ Pasa al paso 3 (Ya es **2NF**).
3. **¿Hay atributos no-clave que dependan de otros no-clave? (Transitividad)**
   - SÍ $\rightarrow$ Se queda en **2NF**.
   - NO $\rightarrow$ Ya es **3NF**. (Generalmente esto es suficiente para aprobar, a menos que te pidan explícitamente BCNF).
4. **(Solo si piden BCNF)**: ¿Hay algún atributo que determine parte de la clave primaria pero que no sea clave él mismo?
   - SÍ $\rightarrow$ Se queda en **3NF**.
   - NO $\rightarrow$ Es **BCNF**.

Guía de Estudio Maestra: SQL, Funciones y Lógica de Conjuntos

Este documento es una expansión exhaustiva diseñada para cubrir no solo la sintaxis, sino la **lógica interna** del motor SQL. Entender *cómo* piensa la base de datos es la única forma de resolver las preguntas trampa de los exámenes.

1. Estructura y Sub-Lenguajes SQL (Análisis Profundo)

En los exámenes teóricos, es común que te pidan clasificar un comando o describir su impacto en la transacción.

DML (Data Manipulation Language) - "Manipulación"

Se encarga de la gestión de los datos *vivos* dentro de las estructuras.

- **Transaccionalidad:** Los comandos DML son transaccionales. Si ejecutas un `UPDATE` y te arrepientes, puedes hacer `ROLLBACK` (si no has hecho commit).
- **SELECT:** Recuperación de datos. Aunque no modifica, se agrupa aquí o en DQL (Data Query Language).
- **INSERT:**
  - *Ejemplo múltiple:* `INSERT INTO alumnos (id, nombre) VALUES (1, 'Ana'), (2, 'Luis');`
- **UPDATE:** Modificación de registros existentes.
  - *Peligro:* Si olvidas el `WHERE`, actualizas **toda** la tabla.
- **DELETE:** Borrado físico de filas.
  - *Detalle:* Registra cada fila borrada en el log de transacciones (es lento en tablas grandes).

DDL (Data Definition Language) - "Definición"

Define y modifica la estructura (metadatos) de la base de datos.

- **Auto-Commit:** La mayoría de bases de datos (como Oracle o SQL Server por defecto) ejecutan un `COMMIT` implícito antes y después de un comando DDL. No se pueden deshacer fácilmente.
- **CREATE:** Crea objetos (tablas, vistas, índices, procedimientos).
  - *Ejemplo:* `CREATE VIEW resumen_ventas AS SELECT ...`
- **ALTER:** Modifica estructuras existentes sin perder datos (generalmente).
  - *Ejemplo:* `ALTER TABLE cursos ADD CONSTRAINT fk_profe FOREIGN KEY ...`
- **TRUNCATE:** A menudo confundido con DML, es técnicamente DDL en muchos motores.
  - *Diferencia clave con DELETE:* `TRUNCATE` reinicia los contadores de identidad (autoincrementales) y es mínimamente logueado, por lo que es muchísimo más rápido pero irreversible.

DCL (Data Control Language) - "Seguridad"

- **GRANT:** Otorga permisos específicos (`SELECT`, `EXECUTE`) sobre objetos específicos a usuarios o roles.

- **REVOKE:** Retira los permisos previamente otorgados.

TCL (Transaction Control Language) - "Integridad"

Controla el "todo o nada" de las operaciones lógicas de negocio.

- **COMMIT:** Confirma permanentemente los cambios de la transacción actual.
- **ROLLBACK:** Restaura el estado de los datos al último `COMMIT`.
- **SAVEPOINT:** Crea un punto intermedio dentro de una transacción al que se puede volver sin deshacer todo.

2. Anatomía de la Sentencia SELECT y el Orden de Ejecución

El error número uno en los exámenes de SQL es intentar usar un alias definido en el `SELECT` dentro del `WHERE`. Esto falla porque el `WHERE` ocurre en el pasado para el motor.

El Orden Lógico Detallado

El motor construye el resultado paso a paso, reduciendo el conjunto de datos en cada etapa:

1. **FROM / JOIN (Producto Cartesiano y Enlace):**
   - El motor identifica las tablas requeridas.
   - Si hay JOINs, primero hace un producto cartesiano (todas las combinaciones) y luego filtra según la cláusula `ON`.
   - *Resultado:* Una "Tabla Virtual 1" con todas las columnas de las tablas involucradas.
2. **WHERE (Filtrado de Filas):**
   - Pasa fila por fila de la Tabla Virtual 1 y evalúa la condición. Si es `FALSE` o `NULL`, la fila se descarta.
   - *Limitación:* No puede ver alias creados en el futuro (`SELECT`) ni resultados de agregaciones (`SUM`).
3. **GROUP BY (Agrupación):**
   - Colapsa las filas restantes en "cubos" basados en los valores de las columnas agrupadas.
   - *Cambio de paradigma:* A partir de aquí, ya no existen filas individuales, solo grupos. Por eso no puedes hacer `SELECT *`.
4. **HAVING (Filtrado de Grupos):**
   - Evalúa condiciones sobre los "cubos" creados.
   - *Uso:* `HAVING SUM(ventas) > 1000`. Esto no se podía hacer en el `WHERE` porque la suma no existía antes de agrupar.
5. **SELECT (Proyección y Cálculo):**
   - Ahora el motor calcula las expresiones (`precio * 1.21`), evalúa funciones y asigna los **Alias** (`AS total_iva`).
   - Si hay `DISTINCT`, elimina duplicados aquí.
6. **ORDER BY (Ordenamiento):**
   - Ordena el resultado final.

- *Curiosidad:* Es el único lugar donde puedes usar los alias del `SELECT` de forma segura, porque ocurre después.

7. **LIMIT / OFFSET / TOP (Paginación):**
   - Corta el número de filas enviadas al cliente.

3. Funciones SQL: Herramientas de Transformación

A. Funciones de Agregación Avanzadas

Operan sobre un conjunto de filas y devuelven un único valor escalar.

Función	Matiz Importante	Ejemplo Trampa
COUNT(*)	Cuenta filas físicas. No le importa el contenido.	Si una tabla tiene 5 filas con todo `NULL`, `COUNT(*)` devuelve 5.
COUNT(col)	Cuenta valores **NO NULL**.	Si de 5 filas, 2 tienen email `NULL`, `COUNT(email)` devuelve 3.
COUNT(DISTINCT col)	Cuenta valores únicos no nulos.	`SELECT COUNT(DISTINCT pais) FROM clientes;`
SUM(col)	Ignora nulos. Si todos son nulos, devuelve `NULL` (no cero).	`SUM(salario)` devuelve `NULL` si no hay salarios, cuidado con sumar esto a otro número.
AVG(col)	Matemáticamente es `SUM(col) / COUNT(col)`.	Si tienes `[10, NULL, 20]`, el promedio es `(10+20)/2 = 15` (ignora el nulo), no `30/3 = 10`.

B. Funciones Escalares y Lógica Condicional

Manejo de Nulos (Lógica Ternaria): En SQL, una expresión booleana puede ser `TRUE`, `FALSE` o `UNKNOWN`.

- **IS NULL / IS NOT NULL:** Única forma correcta de verificar nulos.
- **COALESCE(v1, v2, ... vn):** Devuelve el primer valor no nulo. Es estándar ANSI y muy útil para reportes.

```
SELECT producto, COALESCE(precio_oferta, precio_lista, 0) as precio_final FROM item
```

- **NULLIF(v1, v2):** Devuelve `NULL` si `v1 == v2`. Útil para evitar divisiones por cero (`v1 / NULLIF(v2, 0)`).

Transformación de Datos:

- **String:** `UPPER()`, `LOWER()`, `LENGTH()` (o `LEN`), `SUBSTRING()`, `TRIM()`, `CONCAT()`.
- **Fechas:** `DATEDIFF()` (diferencia entre fechas), `DATE_ADD()` (sumar días/meses), `EXTRACT(YEAR FROM fecha)`.

- **Casting:** `CAST(valor AS tipo)` o `CONVERT()`. Vital cuando quieres concatenar un número con un texto.

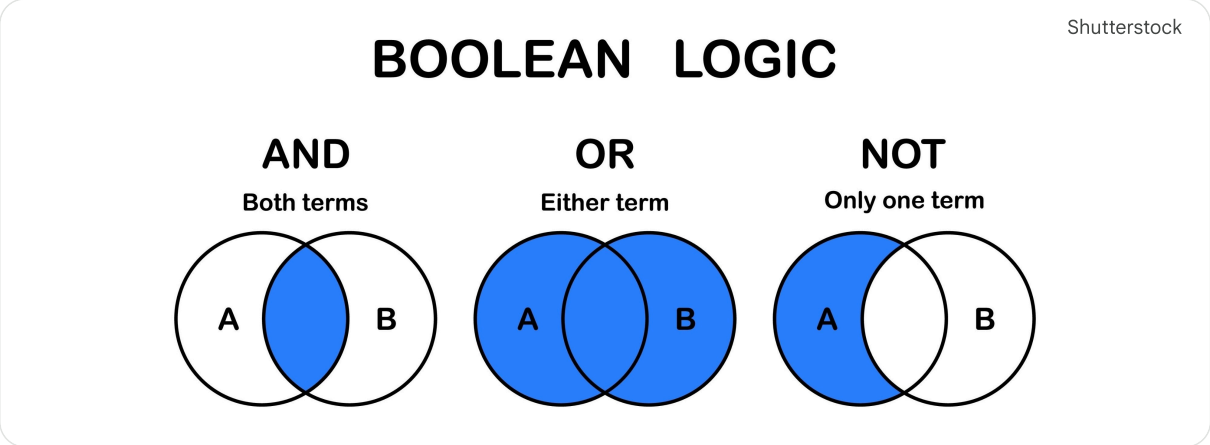
Lógica Condicional (CASE WHEN): Permite lógica compleja fila por fila.

```
SELECT nombre,
    CASE
        WHEN saldo > 1000 THEN 'VIP'
        WHEN saldo < 0 THEN 'Deudor'
        ELSE 'Regular'
    END as categoria_cliente
FROM cuentas;
```

4. Los JOINS: Dominando la Teoría de Conjuntos

Los JOINs no solo "unen tablas", sino que crean nuevos conjuntos de datos basados en relaciones lógicas.

INNER JOIN (La Intersección Estricta)



Shutterstock

- **Lógica:** $A \cap B$. Solo filas donde la condición `ON` es verdadera en ambos lados.
- **Efecto de filtrado:** Funciona como un filtro. Si enlazas una tabla de 1 millón de filas con una de 10 filas mediante INNER JOIN, el resultado tendrá como máximo 10 filas (o más si hay duplicados en la relación 1:N).
- **Código:**

```
SELECT e.nombre, d.departamento
FROM Empleados e
INNER JOIN Departamentos d ON e.depto_id = d.id;
```

LEFT JOIN (La Preservación Izquierda)

- **Lógica:** $(A - B) \cup (A \cap B)$. Todas las filas de la tabla izquierda (la primera en el `FROM`), acompañadas de los datos de la derecha si existen, o `NULL` si no.
- **Uso Crítico:** Detección de datos faltantes. "Clientes que NUNCA han comprado".
- **Código de Detección:**

```
SELECT c.nombre
FROM Clientes c
LEFT JOIN Pedidos p ON c.id = p.cliente_id
WHERE p.id IS NULL; -- Esto filtra y deja solo los que NO cruzaron.
```

RIGHT JOIN (La Preservación Derecha)

- **Lógica:** Inverso al Left Join. Preserva la tabla de la derecha.
- **Nota Práctica:** Rara vez se usa en la práctica porque leer de izquierda a derecha (`LEFT JOIN`) es más natural para la lectura occidental.

FULL OUTER JOIN (La Unión Total)

- **Lógica:** $A \cup B$. Todo lo de A y todo lo de B. Si coinciden, se unen. Si no, se rellenan con NULLs los huecos.
- **Soporte:** No todos los motores lo soportan (MySQL no lo tiene nativo, se simula con `LEFT JOIN` UNION `RIGHT JOIN`).

CROSS JOIN (El Producto Cartesiano)

- **Lógica:** $A \times B$. Combina cada fila de A con todas las filas de B.
- **Riesgo:** Genera un crecimiento exponencial de filas. 100 filas x 100 filas = 10,000 filas.
- **Uso Legítimo:** Generar combinaciones masivas (ej: "Para cada talla, generar todos los colores disponibles").

```
SELECT t.talla, c.color
FROM Tallas t CROSS JOIN Colores c;
```

SELF JOIN (Auto-referencia)

Es un join de una tabla consigo misma. Requiere obligatoriamente el uso de **Alias** para distinguir la "versión 1" de la "versión 2" de la tabla.

- **Ejemplo Clásico:** Jerarquías (Empleados y Jefes).

```
SELECT e.nombre AS empleado, j.nombre AS jefe
FROM Empleados e
LEFT JOIN Empleados j ON e.jefe_id = j.id; -- 'j' es la misma tabla pero actuando c
```

5. Escenarios "What If..." y Preguntas Avanzadas

Estas preguntas evalúan tu capacidad de predecir el comportamiento del motor ante situaciones límite.

P: ¿Qué pasa si uso `UNION` en lugar de `UNION ALL` ?

- **R:**

- `UNION` : Combina los resultados y **elimina duplicados**. Implica un costo de rendimiento porque la base de datos debe ordenar y comparar para desduplicar.
- `UNION ALL` : Combina los resultados "tal cual", manteniendo duplicados. Es mucho más rápido.
- *Consejo:* Si sabes que los datos no se solapan, usa siempre `UNION ALL` por rendimiento.

P: ¿Qué pasa si hago un Join con una condición `NULL = NULL` ?

- **Escenario:** Tienes dos tablas y ambas tienen `NULL` en la columna de unión.
- **R:** Las filas **NO** se unirán.
- *Razón:* Como `NULL != NULL` (es desconocido), la condición del `ON` falla. Si necesitas unir nulos, debes usar `ON COALESCE(a.col, 0) = COALESCE(b.col, 0)` o funciones específicas de manejo de nulos.

P: ¿Qué sucede si filtro en el `WHERE` una columna que usé para agrupar en el `GROUP BY` ?

- **R:** Es válido y correcto.
- *Lógica:* El `WHERE` filtra los datos crudos antes de que entren al cubo de agrupación. Ejemplo: "Agrupar ventas por ciudad, pero solo considerar las ventas de productos electrónicos".

```
SELECT ciudad, SUM(ventas)
FROM tabla
WHERE categoria = 'Electronica' -- Filtra ANTES
GROUP BY ciudad;
```

P: ¿Qué pasa si selecciono una columna que NO está en el `GROUP BY` y NO tiene función de agregación?

- **Código:** `SELECT ciudad, nombre_cliente, SUM(ventas) FROM tabla GROUP BY ciudad;`
- **R:**
  - En modo estricto (SQL Server, Oracle, MySQL moderno con `ONLY_FULL_GROUP_BY`): **ERROR**. No puedes pedir "nombre_cliente" si has agrupado por ciudad, porque hay muchos clientes en una ciudad y el motor no sabe cuál mostrar.
  - En MySQL antiguo: Devuelve un valor **aleatorio/indeterminado** de esa columna para el grupo. ¡Muy peligroso!

P: ¿Qué pasa si sumo un número con un NULL? (`SELECT 100 + NULL`)

- **R:** El resultado es `NULL` .
- *Concepto:* El `NULL` es "contagioso" en operaciones aritméticas. Una sola celda nula puede arruinar un cálculo de reporte entero si no se trata con `COALESCE` o `ISNULL` .

P: ¿Qué pasa si hago un `DELETE` sin `WHERE` vs un `TRUNCATE` ?

- **R:** Ambos borran todos los datos.
  - `DELETE` va fila por fila, dispara *Triggers*, guarda logs y es lento. Se puede revertir dentro de una transacción.