Perfecto Don Daniel. ¡Vargas Cipollo!

Te prepararé unos apuntes optimizados para repasar con rapidez y claridad, abarcando los cinco subtemas del Tema 4: Wrappers, Descripción de fuentes de datos, String Matching, Schema Matching e Data Matching.

Organizaré el contenido en listas de puntos clave, evitando párrafos extensos. Al final de cada subtema, incluiré ejemplos de ejercicios tipo examen con sus soluciones si los documentos lo permiten.

En cuanto estén listos, te aviso para que puedas empezar el repaso cuanto antes.

Tema 4: Integración de Datos – Apuntes de Repaso (Examen)

1. Wrappers

- **Definición:** Programas *intermediarios* que traducen las consultas del sistema integrador al lenguaje/formato de una fuente de datos heterogénea, y luego convierten los resultados de la fuente al formato común del sistema integrado. Son esencialmente "*envoltorios*" que ocultan las peculiaridades de cada fuente.

- **Funciones principales de un wrapper:**

    - Conectar con la fuente de datos específica (BD, servicio web, archivo, etc.) usando sus protocolos o lenguaje nativo.
    - **Traducir** la consulta global a una consulta entendible por la fuente (por ejemplo, transformar una consulta SQL genérica a una consulta SQL específica de una base Oracle, o a una API web).
    - **Extraer** o recibir los datos de la fuente y **transformarlos** al esquema y formato global (p. ej., convertir resultados JSON/XML/CSV de la fuente al formato canónico del integrador).
    - Opcionalmente, **filtrar y limpiar** datos de la fuente (eliminar campos innecesarios, estandarizar unidades o codificaciones) antes de enviarlos al integrador.

- **Ventajas:**

    - Permiten al sistema de integración acceder a fuentes heterogéneas de manera transparente y uniforme, sin tener que programar la lógica de acceso cada vez.
    - Aíslan cambios: si cambia una fuente (estructura o modo de acceso), solo se actualiza el wrapper correspondiente, evitando modificar el resto del sistema.
    - Pueden reutilizarse componentes de wrappers para fuentes similares, reduciendo esfuerzo en integraciones futuras.

- **Desventajas:**

    - Desarrollo costoso y específico: normalmente se debe construir un wrapper por cada fuente (proceso manual que requiere conocer bien la fuente).
    - Fragilidad ante cambios: si la fuente de datos cambia su formato, estructura o interfaz, el wrapper puede dejar de funcionar hasta ser modificado.
    - Mantenimiento complejo cuando hay muchas fuentes: administrar y actualizar numerosos wrappers se vuelve laborioso.

- *Arquitectura típica:* **Fuente de datos ⟶ Wrapper ⟶ Mediador/Integrador**. El mediador envía consultas globales al wrapper, el wrapper las ejecuta en la fuente y devuelve los datos integrados al mediador.

- **Tipos de wrappers:**

    - **Construcción manual:** Un programador escribe el wrapper a medida (en un lenguaje como Java, Python, SQL, XSLT, etc.), controlando exactamente cómo conectarse y mapear los datos. Garantiza alta precisión pero requiere mucho esfuerzo experto.
    - **Generación automática o semiautomática:** Uso de herramientas o algoritmos de aprendizaje que inducen wrappers a partir de ejemplos (común en extracción de datos web). Ahorra tiempo pero suele requerir un conjunto de entrenamiento y puede no lograr cubrir todos los casos.

- **Ejemplo (tipo examen):** *¿Cuál es la función de un wrapper en un sistema de integración de datos y qué ventajas aporta?*

- **Solución:** Un wrapper actúa como traductor entre el integrador y una fuente específica. Permite que el integrador consulte una fuente heterogénea como si fuera homogénea. **Ventajas:** oculta la complejidad de fuentes heterogéneas, facilita añadir nuevas fuentes sin cambiar el núcleo del sistema, y aísla los cambios de las fuentes (si una fuente cambia, se actualiza solo su wrapper).

2. Descripción de fuentes de datos (GAV, LAV, GLAV)

- En la integración de datos es necesario **describir las fuentes** y cómo se relacionan con el esquema global. Esto se hace mediante *mapeos* que conectan elementos de las bases locales con elementos del esquema unificado. Existen enfoques clásicos para estos mapeos: **Global-As-View (GAV)**, **Local-As-View (LAV)** y el híbrido **Global-Local-As-View (GLAV)**.

- **Enfoque GAV (Global as View – "Global como Vista"):** El **esquema global** se define **como vistas sobre las fuentes locales**. Es decir, cada relación/tabla global se expresa mediante una consulta que combina una o varias fuentes de datos. El integrador "sabe" exactamente cómo obtener cada dato global de las fuentes. *Ejemplo:* una vista global `Clientes` podría definirse como `SELECT nombre, tel FROM Fuente1.NombreTel JOIN Fuente2.Teléfonos`.

    - *Ventaja:* Las consultas globales se responden fácilmente *desplegando* (expandiendo) estas vistas predefinidas. Simplifica el **procesamiento de consultas**, ya que el camino hacia los datos locales está especificado.
    - *Desventaja:* **Poco flexible ante cambios** – si se agrega una nueva fuente o cambia una existente, hay que modificar las definiciones globales afectadas. Mantenimiento costoso porque el esquema global depende directamente de las fuentes.

- **Enfoque LAV (Local as View – "Local como Vista"):** Las **fuentes de datos** se describen **como vistas del esquema global**. Aquí el esquema global es independiente, y para cada fuente se indica cómo sus datos se corresponden con consultas o vistas en términos del esquema global. *Ejemplo:* una fuente local `ClientesCRM` se describe con una sentencia del estilo "`ClientesCRM(nombre, tel) es vista de Global.Cliente(nombre, telefono)` filtrando país='ES'".

    - *Ventaja:* **Facilidad de extensión** – se pueden agregar nuevas fuentes sin tocar el esquema global; basta con definir su vista/mapeo. El esquema global permanece estable, centralizando la semántica común.

    - *Desventaja:* **Consultas globales más complejas de resolver** – el sistema de integración debe **razonar/planificar** cómo recombinar las vistas locales para responder una consulta. Requiere un motor de inferencia (resolver consultas mediante reglas inversas), lo cual es más complejo y costoso en tiempo.

- **Enfoque GLAV (Global-Local as View):** Es un enfoque generalizado que combina GAV y LAV. Permite mapeos **mucho más expresivos** donde partes del esquema global se mapean a partes de fuentes y viceversa en forma de correspondencias lógicas (assertions). En la práctica, GLAV abarca casos que no se pueden expresar puramente con GAV o LAV.

    - GLAV suele utilizarse en sistemas modernos de integración basados en **correspondencias lógicas** (p. ej., reglas en lenguajes como Datalog o RDF mappings). **Ventaja:** mayor flexibilidad para modelar relaciones complejas; **Desventaja:** la inferencia y el cálculo de consultas pueden ser incluso más complicados.

- **Comparativa GAV vs LAV:**

    - GAV fija las reglas de obtención de datos en el diseño del esquema global (simple de consultar, difícil de mantener).
    - LAV delega la carga al momento de la consulta (flexible para mantener, pero complejiza la respuesta de consultas).
    - GLAV busca un equilibrio, permitiendo mappings muchos-a-muchos entre global y local, pero requiriendo algoritmos sofisticados de consulta.

- **Ejemplo (tipo examen):** *Menciona dos diferencias entre los enfoques GAV y LAV en la descripción de fuentes de datos.*

- **Solución: GAV:** el esquema global se define en función de las fuentes; es sencillo responder consultas pero costoso agregar o modificar fuentes. **LAV:** las fuentes se definen en función del esquema global; es fácil añadir nuevas fuentes definiendo sus vistas, pero responder las consultas globales requiere un proceso de razonamiento más complejo por parte del integrador.

3. String Matching (Emparejamiento de cadenas)

- **Definición:** Técnicas para *comparar cadenas de texto* y determinar su grado de similitud o diferencia. En integración de datos, el string matching se usa para reconocer cuando dos textos representan el mismo dato o entidad (por ejemplo, dos nombres ligeramente distintos que corresponden a la misma persona, o dos etiquetas de columna con significado equivalente).

- **Aplicación:** Es fundamental para resolver problemas de *heterogeneidad lexical* – diferencias en escritura, tipografía, abreviaturas o errores ortográficos. Permite identificar coincidencias pese a variaciones: por ejemplo, “IBM” vs “I.B.M.”, “García” vs “Garcia” (sin tilde), o diferentes formatos de una dirección.

- **Principales técnicas de comparación de cadenas:** *(cada técnica ofrece una medida de similitud/distancia y tiene distintas propiedades)*

    - **Distancia de Levenshtein (distancia de edición):** Número mínimo de operaciones de edición (inserciones, borrados o sustituciones de caracteres) para transformar una cadena en otra. Útil

para capturar errores tipográficos *pequeños*. Por ejemplo, convertir **"casa"** en **"caza"** requiere 1 sustitución (s→z), entonces Levenshtein = 1 (baja distancia implica cadenas muy similares).
    - **Distancia de Jaro-Winkler:** Métrica especializada para comparar *nombres cortos* (personas, empresas, etc.), considerando transposiciones de caracteres y prefijos comunes. Devuelve un índice de 0 a 1 (1 = cadenas idénticas). Es efectiva para coincidencias en nombres con pequeñas permutaciones o errores (ej.: "MARÍA" vs "MARIA" con tilde).
    - **Coeficiente de Jaccard (sobre q-gramas o palabras):** Mide similitud como el tamaño de la intersección dividido por el de la unión de conjuntos de fragmentos (subcadenas) o palabras. Útil para cadenas más largas o comparaciones de *conjuntos de palabras*. Por ejemplo, para cadenas tratadas como conjuntos de 3-letras. Si comparten muchos trigramas, la similitud de Jaccard será alta.
    - **Similitud coseno (Vectores TF-IDF):** Representa las cadenas como vectores en un espacio dimensional (por ejemplo, basado en términos o *q-gramas*) y calcula el coseno del ángulo entre ellos. Es útil cuando se comparan textos más extensos (nombres compuestos, descripciones) considerando frecuencia de términos, mitigando impacto de palabras comunes.
    - **Algoritmos fonéticos (ej.: Soundex):** Convierten las palabras a un código que representa su sonido. Dos cadenas tienen el mismo código Soundex si *suenan* similar en inglés/español. Ayuda a emparejar variantes ortográficas de nombres propios que se pronuncian parecido (ej.: "Lewis" vs "Luis").

- **Preprocesamiento:** Antes de comparar, se suele **normalizar** las cadenas: pasar a minúsculas, eliminar acentos/puntuación, expandir abreviaturas ("St." → "Street"/"San"), eliminar espacios extra, etc. Esto aumenta la eficacia de las técnicas de matching al eliminar diferencias superficiales.

- **Uso combinado:** En la práctica, varias métricas pueden combinarse o usarse secuencialmente. Por ejemplo, primero usar Soundex para filtrar candidatos por sonido y luego Levenshtein para calcular diferencias exactas. También se establecen **umbrales**: p. ej., considerar "match" aquellas cadenas con similitud de Jaro-Winkler ≥ 0.85.

- **Ventajas/Desventajas:**

    - **Ventaja:** Permite detectar equivalencias de cadenas que no son exactamente iguales, mejorando la integración (menos datos duplicados bajo nombres distintos). Cada técnica está especializada: unas son sensibles a errores de tipeo (Levenshtein), otras a permutaciones (Jaro-Winkler), otras a diferencias de tokens (Jaccard, Coseno).
    - **Desventaja:** No existe una métrica perfecta universal. Algunas pueden dar *falsos positivos* (considerar similares cadenas que en contexto no lo son) o *falsos negativos* (no identificar como match dos cadenas semánticamente iguales pero muy distintas en texto). A menudo se requiere ajustar umbrales o combinar métodos, y el cálculo de similitud puede ser costoso si hay muchísimos datos que comparar.

- **Ejemplo (tipo examen):** *Calcular la distancia de Levenshtein entre **"casa"** y **"caza"**.*

- **Solución:** Se substituye la **'s'** por **'z'**, que es **1 operación** de edición. Por lo tanto, la distancia de Levenshtein es **1** (muy baja, indicando que las cadenas difieren en un solo carácter).

4. Schema Matching (Integración de esquemas)

- **Definición:** Proceso de *emparejar elementos de distintos esquemas* (atributos, tablas o campos) que tienen el mismo significado semántico en diferentes fuentes de datos. Es decir, identificar qué columna o campo de la Base de Datos A corresponde (representa la misma información) que un campo de la Base B, incluso si tienen nombres o estructuras diferentes. El resultado del schema matching suele ser un conjunto de **correspondencias** (mapeos) entre atributos de esquemas distintos.

- **Importancia:** Es un paso clave para fusionar esquemas en una integración de datos o para migración de bases, ya que permite crear un **esquema global unificado**. Debido a que distintas fuentes pueden etiquetar y organizar datos de forma heterogénea, el schema matching ayuda a *resolver heterogeneidades estructurales y de nomenclatura*.

- **Dificultad:** Muchas veces no hay una forma trivial de saber si dos campos coinciden, especialmente si los nombres no son idénticos. La tarea puede requerir *conocimiento del dominio* o algoritmos inteligentes, y aunque se automatiza parcialmente, suele necesitar validación humana.

- **Criterios comunes para comparar elementos de esquema:** *(se pueden emplear múltiples criterios en conjunto para mejorar la precisión)*

    - **Similitud léxica (de nombres):** Compara las cadenas que conforman los nombres de tablas/atributos. Usa técnicas de string matching (como las mencionadas) para detectar nombres parecidos o relacionados (ej.: "cod_postal" vs "CP" vs "zipcode" pueden indicar código postal). Incluye uso de sinónimos, diccionarios o ontologías básicas para reconocer términos equivalentes.
    - **Similitud estructural:** Compara la posición o rol de un elemento en el esquema. Por ejemplo, si dos tablas en diferentes BD tienen la misma relación jerárquica o llaves similares. Si *Cliente* en un esquema tiene una sub-entidad *Dirección*, y en otro esquema *Customer* tiene *Address*, esa correspondencia de contexto refuerza que "Cliente=Customer" y "Dirección=Address". También considera tipos de datos (un campo numérico difícilmente coincide con uno texto si sus tipos no concuerdan).
    - **Similitud basada en datos (instancias):** Compara los *valores contenidos* en los campos. Si dos columnas en distintas fuentes comparten muchos valores o patrones (por ej., ambas contienen cadenas con formato de correo electrónico, o ambas listas de apellidos comunes), es indicio de correspondencia. Permite detectar matches incluso cuando los nombres difieren significativamente, basándose en la distribución de datos.
    - **Similitud semántica:** Utiliza conocimiento externo (ontologías, catálogos de sinónimos, taxonomías del dominio) para determinar si dos elementos representan el mismo concepto. Por ejemplo, saber que "empleado" y "trabajador" son sinónimos, o que "ProductoID" es lo mismo que "IdArticulo" mediante un vocabulario común.

- **Herramientas y técnicas:** Existen algoritmos y sistemas de schema matching que combinan los criterios anteriores. Pueden asignar **pesos** a cada criterio y calcular un puntaje total de similitud entre posibles pares de atributos. Algunos emplean *aprendizaje automático* entrenado con ejemplos de correspondencias correctas. Finalmente, suelen aplicar un **umbral** para proponer correspondencias (ej.: si similitud > 0.8) o rankear las más probables.

- **Intervención manual:** Dado que los métodos automáticos no garantizan 100% de acierto, normalmente un experto revisa y ajusta las correspondencias propuestas. Un buen proceso de schema

matching equilibra la velocidad de algoritmos automáticos con la *validación humana* para los casos dudosos.

- **Ventajas:** Automatizar en parte el schema matching ahorra muchísimo esfuerzo en grandes esquemas y puede descubrir coincidencias no obvias. Usar múltiples criterios mejora la calidad de los resultados (combina evidencia léxica, estructural, etc.). Un esquema global bien alineado permite integrar datos con menos inconsistencias.

- **Desventajas:** La automatización puede producir *falsos matches* (atributos emparejados incorrectamente) que, si no se corrigen, introducen errores en la integración. El proceso puede ser complejo (NP-completo si se considera la búsqueda exhaustiva de alineaciones). Requiere mantenimiento: si los esquemas cambian, hay que re-ejecutar o ajustar el matching. Además, desarrollar y ajustar estos algoritmos es en sí un desafío. En muchos casos, **se necesita intervención humana**, lo cual consume tiempo, especialmente si el algoritmo arroja muchos candidatos a revisar.

- **Ejemplo (tipo examen):** *Dadas las siguientes dos bases de datos, empareja los campos equivalentes:*

    - *Esquema A:* {**Nombre**, **Apellido**, **Teléfono**}
    - *Esquema B:* {**Name**, **Phone**, **Surname**}

- **Solución:** Las correspondencias identificadas serían: **Nombre = Name**, **Apellido = Surname**, **Teléfono = Phone**. (Aquí la similitud léxica y de significado permite emparejar correctamente cada campo). En un caso real, podría haber campos menos obvios, pero se aplicaría análisis de nombres, estructura y datos para determinarlos.

5. Data Matching (Emparejamiento de datos)

- **Definición:** Proceso de identificar registros que se refieren a la *misma entidad del mundo real* en uno o más conjuntos de datos. También llamado **record linkage** o **deduplicación**. Consiste en comparar campos de distintos registros y decidir si corresponden al mismo individuo/objeto a pesar de posibles diferencias o variaciones en los datos.

- **Objetivo:** *Fusionar o eliminar duplicados* y unificar información de una entidad. En integración de datos, el data matching permite combinar datos de distintas fuentes referentes al mismo cliente, producto, etc., obteniendo un conjunto único sin redundancias (o enlazando registros equivalentes entre sistemas).

- **Retos:** Los identificadores únicos pueden no existir o no coincidir entre fuentes, por lo que se usan atributos descriptivos (nombre, fecha de nacimiento, dirección, códigos, etc.) para comparar. Estos pueden contener errores, distintas convenciones (ej. “Juan Pérez” vs “Perez, Juan”), lo que hace necesario usar **comparaciones aproximadas** (string matching en nombres, tolerancia en fechas/números) en lugar de solo comparaciones exactas.

- **Pasos típicos en un proceso de Data Matching:**

    - **Preprocesamiento y normalización:** limpiar los datos para hacer comparaciones justas. Ejemplo: estandarizar mayúsculas/minúsculas, quitar espacios/guiones en números de teléfono, unificar formatos de fecha, etc. También seleccionar qué campos serán utilizados para el matching (campos clave).

- **Bloqueo o indexación:** reducir el universo de comparaciones posibles agrupando registros por alguna clave aproximada. Por ejemplo, comparar solo registros que tengan la misma inicial del apellido o el mismo código postal. Esto evita comparar cada registro contra todos (lo cual escala cuadráticamente y sería inviable con muchos datos). **Ventaja:** mejora eficiencia; **Riesgo:** si el criterio de bloqueo es muy estricto, puede dejar fuera pares que eran matches verdaderos.
- **Comparación de registros:** para cada par de registros candidato (dentro del mismo bloque, por ejemplo), calcular **medidas de similitud** en sus campos. Se aplican técnicas de string matching en nombres/textos, comparación numérica en valores numéricos (ej. diferencia absoluta de edades), comparaciones exactas en identificadores únicos si existen, etc. Se suele obtener varios puntajes (uno por campo comparado).
- **Decisión de correspondencia:** integrar las evidencias de comparación para determinar si es el mismo elemento. Puede usarse un **modelo de puntaje** (por ejemplo, sumar ponderadamente similitudes de cada campo) o un algoritmo de *machine learning* entrenado con ejemplos de match/no-match. Comúnmente se define un **umbral**: si el puntaje total ≥ T, se considera *match* (misma entidad); si es muy bajo, *no match*; si queda en zona intermedia, puede marcarse como *posible match* para revisión manual.
- **Fusión y consolidación:** (si es un proceso de integración) una vez identificados los duplicados o correspondencias, se fusionan o conectan los datos. Por ejemplo, si dos registros de distintas fuentes resultan ser la misma persona, se puede crear un registro maestro unificado que combine los atributos (tomando los más actualizados o confiables de cada fuente). Alternativamente, se marca un ID común para referenciarlos como la misma entidad.

- **Técnicas y enfoques:**

  - **Determinístico (basado en reglas):** Se definen reglas explícitas para considerar dos registros iguales. Ej: “Coinciden DNI o correo electrónico => mismo individuo” o “Mismo apellido y misma fecha de nacimiento => posible match”. Es simple de entender y controlar, pero no cubre casos con datos levemente diferentes.
  - **Probabilístico/estadístico:** Basado en modelos como el de Fellegi-Sunter, que asignan probabilidades a la concordancia de cada campo y calculan la probabilidad global de match. Requiere entrenar o estimar probabilidades *m* (de que un campo coincida cuando es mismo entidad) vs *u* (coincidencia por azar). Más flexible y suele lograr mayor *recall* (encuentra más verdaderos matches) pero puede generar falsos positivos si no está bien calibrado.
  - **Aprendizaje automático:** Entrenar clasificadores (árboles de decisión, SVM, redes neuronales) con ejemplos de pares “mismo” vs “distinto”. El modelo aprende patrones complejos de similitud. Puede lograr alta precisión si hay datos de entrenamiento de calidad, pero es complejo de preparar y explicar.

- **Ventajas:** Un buen data matching mejora la **calidad de los datos integrados**, eliminando duplicados y agregando información dispersa de la misma entidad. Automatizarlo ahorra la revisión manual de miles de registros. Es esencial en áreas como limpieza de datos, consolidación de clientes, detección de fraude (identificar cuando una persona tiene varias identidades, etc.).

- **Desventajas:** Ningún método garantiza 100% de acierto: siempre hay un compromiso entre *falsos negativos* (no detectar duplicados existentes) y *falsos positivos* (unir registros que en realidad son distintos). Configurar umbrales adecuados es difícil y a menudo se necesita *supervisión humana* para revisar casos dudosos. Además, el proceso puede ser costoso en cómputo cuando los datasets son muy grandes (incluso con bloqueo, algunas comparaciones son intensivas debido a algoritmos de string

matching). La implementación y mantenimiento de reglas o modelos de matching requieren conocimiento experto del dominio de datos.

- **Ejemplo (tipo examen):** *Tenemos dos registros de clientes de diferentes fuentes:*
    - *Registro A:* Nombre: **"María López"**, Edad: **23**, Ciudad: **"Barcelona"**.
    - *Registro B:* Nombre: **"M. Lopez"**, Edad: **23**, Ciudad: **"BARCELONA"**. *¿El sistema debería considerarlos la misma persona? ¿Por qué?*

- **Solución:** Sí, probablemente representan la **misma persona**. La comparación de campos muestra alta similitud: el nombre "María López" vs "M. Lopez" coincide en apellido *López* completo y el nombre de pila "M." puede ser **María** abreviado (además, **Soundex** u otro algoritmo fonético coincidiría para "María" vs "M."). La edad es exactamente la misma (23) y la ciudad, tras normalizar mayúsculas/acentos, también coincide ("Barcelona"). Dado que todos los campos clave coinciden o son muy similares, el algoritmo de matching sumaría suficiente evidencia (sobre un umbral establecido) para vincularlos como el mismo cliente.