Unidad 1: “Grafos”

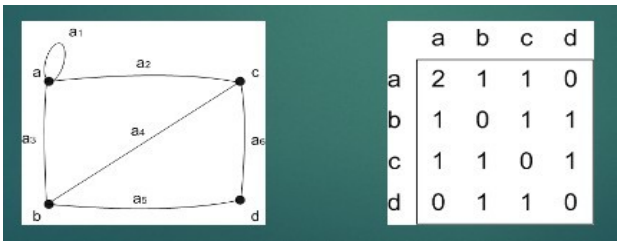
Un grafo es una pareja G = (V, A), donde V es un conjunto de puntos, llamados vértices, y A es un conjunto de pares de vértices, llamadas aristas. Grafo es conjunto de vértices (nodos) y arcos que se relacionan.

Representación Computacional Estática

Se establece un espacio fijo, el cual no cambia en función de cómo cambia el grafo. Desde un principio contempla todas las relaciones posibles. Su representación computacional se construye sobre estructuras como vectores y matrices.

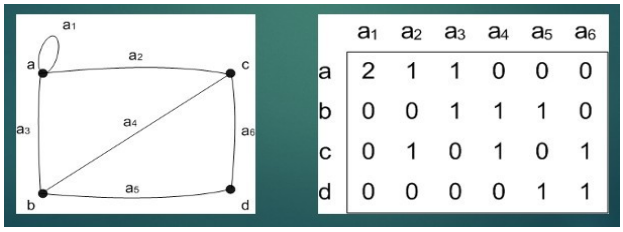
- Para grafos mas pequeños => Accesos rapidos y directos
- Grafos densos (muchas conexiones)
- No se puede modificar

Matriz de adyacencia: se asocia cada fila y cada columna a cada nodo del grafo, siendo los elementos de la matriz la relación entre los mismos, tomando como valores 1 si une una arista, 0 sino, y 2 para bucles.



	a	b	c	d
a	2	1	1	0
b	1	0	1	1
c	1	1	0	1
d	0	1	1	0

Matriz de incidencia: se asocia cada fila con un nodo y cada columna con una arista del grafo, siendo los elementos de la matriz la relación un 1 si dicho nodo es incidente con dicha arista, y 0 en caso contrario.



	a_1	a_2	a_3	a_4	a_5	a_6
a	2	1	1	0	0	0
b	0	0	1	1	1	0
c	0	1	0	1	0	1
d	0	0	0	0	1	1

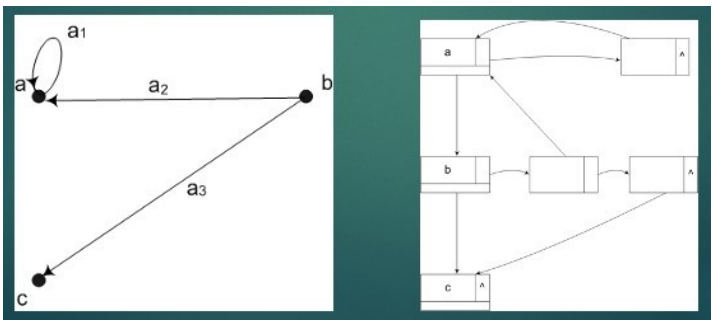
Representación Computacional Dinámica:

El espacio utilizado va cambiando en función de cómo va cambiando el grafo. Para su implementación es necesario el uso de punteros que vinculen las diferentes posiciones de memoria que representan los vértices y los arcos.

- Grafo grande con gran cantidad de vértices O para grafos dispersos
- Si el grafo tiende a cambiar
- Eficiente en memoria
- Mayor complejidad implementacion

• **Listas de adyacencia**: Lista que tiene un nodo por cada vértice. A su vez, cada vértice tiene una sublista con los arcos que salen de él.

-*NODOS TIPO VÉRTICE* tienen tres componentes: uno para sus atributos, otro para apuntar al siguiente nodo de la lista y un tercer puntero a su sublista de aristas.



-*NODOS TIPO ARISTA* tienen dos atributos, un puntero a la arista siguiente y un puntero al vértice al cual dirige esa arista.

Pfaltz: Esta implementación es una ampliación de lo que es la lista de adyacencia. En vez de tener un puntero a una lista de arcos salientes del nodo, vamos a tener dos cosas: un puntero a una lista de arcos salientes y un puntero a una lista de arcos entrantes.

La cantidad de nodos arcos no se van a duplicar, sino que al nodo de tipo arco se le va a agregar un puntero al próximo entrante, al próximo saliente y un puntero al nodo origen

Algoritmo de dijkstra

El algoritmo de dijkstra nos permite calcular la distancia mínima para llegar de un nodo origen a todos los nodos destino que se pueda. Hay dos formas de hacerlo y ambas son recursivas.

- **Búsqueda en profundidad (depth first):** va lo más lejos posible por un camino antes de retroceder. Sirve para explorar, pero no siempre encuentra el camino más corto si hay varios caminos posibles.
- **Búsqueda en anchura (breadth first):** explora primero los vecinos directos del nodo, luego los vecinos de esos, y así sigue. Encuentra el camino con menos pasos, pero no siempre el de menor costo si los caminos tienen distintos valores.

Arboles

Es un grafo en el cual la unicidad se cumple en un solo sentido. Un solo predecesor, pero varios sucesores.

- **Representación Computacional Estática**: un árbol se representa en forma estática a través de un vector.
- **Representación Computacional Dinámica**: un árbol se representa en forma dinámica a través de un conjunto de nodos de igual tipo vinculados entre si a través de punteros o links. Se puede usar lista de adyacencia

Características

- **Completo**: Todos los nodos que no son hojas tienen el mismo grado, excepto posiblemente el último último nivel, el cual es completado de izquierda a derecha.
- **Árbol lleno**: Es un árbol completo y todas sus hojas se encuentran en el mismo nivel.
- **Balanceado**: Todos los subárboles desde la raíz pesan lo mismo, o sea tienen la misma cantidad de elementos.
- **Perfectamente Balanceado**: Cuando esta balanceado en todos sus niveles.

Búsqueda en Arboles

Búsqueda en arboles se realizar por niveles y no por elementos. Necesitamos tantas preguntas como niveles tenga el árbol. En un árbol la búsqueda es logarítmica. Comparado con una estructura de datos básica donde es lineal.

Recorridos/Barridos: Es el proceso de moverse entre los nodos de un árbol

-Recorrido en profundidad; son definidos en forma recursiva

- **Preorden**: Nodo se lee apenas se llega al mismo, puede aplicarse a árboles no binarios..
- **Postorden**: Nodo se lee cuando se va del mismo y no se va a regresar, puede aplicarse a árboles no binarios.
- **Inorden/Simetrico**: el nodo se lee cuando se cambia de rama en el árbol, sólo para árboles binarios.

-Recorrido en anchura: No son considerados recursivos. Se lee por niveles.

Arbol de búsqueda: Es una estructura que permite **buscar, insertar y eliminar datos** de forma rápida, siguiendo un **orden**. En estos árboles, **los valores se ubican según un criterio**: Los menores van a la izquierda y los mayores a la derecha.

Un **árbol binario de búsqueda (ABB)** cumple esta regla para cada nodo. Si lo recorremos en *inorden*, los valores aparecen ordenados de menor a mayor.

Para que funcione bien, el árbol debe estar **balanceado**, es decir, que las ramas izquierda y derecha tengan una cantidad de nodos o niveles similar (pueden diferir como mucho en 1). Cuando esta condición se cumple en todos los nodos, puede llamarse **perfectamente balanceado** o **AVL** (cuando se mide por niveles).

Algoritmos de rotación a izquierda o derecha de un subárbol dado.

Los algoritmos de rotación sirven para balancear un árbol en caso de estar desbalanceado. Si la futura raíz está a la derecha entonces se rota el árbol a izquierda, pero si la futura raíz está a la izquierda entonces se rota el árbol a la derecha. Buscamos reposicionar los nodos, sin cambiar relaciones de padre-hijo.

Unidad 2: “Métodos de Clasificación/Ordenamiento de valores o datos”

-Bubble Sort: Algoritmo de ordenamiento **iterativo** y sencillo, basado en **comparaciones consecutivas**. Recorre el arreglo varias veces y en cada pasada "burbujea" el mayor (o menor) elemento hacia su posición final. Su complejidad es **$O(n^2)$** en el peor y promedio de los casos**. Pasos:**

- Recorrer el arreglo desde el inicio.
- Comparar cada par de elementos adyacentes.
- Si están en el orden incorrecto, **intercambiarlos**.
- Repetir el proceso, reduciendo el rango de comparación, hasta que no haya más intercambios.

Desventaja: Es **ineficiente** para listas grandes por su alto número de comparaciones e intercambios. Aunque es fácil de implementar, no se recomienda en aplicaciones donde el rendimiento sea importante.

-Selection Sort: Algoritmo de ordenamiento iterativo que divide el arreglo en una parte ordenada y una no ordenada. En cada pasada, selecciona el menor elemento de la parte no ordenada y lo coloca en su posición correcta. Su complejidad también es $O(n^2)$. **Pasos:**

- Buscar el mínimo elemento del arreglo.
- Intercambiarlo con el primer elemento.
- Repetir el proceso con el subarreglo restante (sin el primer elemento ya ordenado).
- Continuar hasta que todos los elementos estén en orden.

Desventaja: Aunque realiza menos intercambios que Bubble Sort, sigue siendo ineficiente para listas grandes por la gran cantidad de comparaciones. Además, no es estable, ya que puede cambiar el orden relativo de elementos iguales.

-Merge Sort: Algoritmo recursivo basado en el paradigma Divide y Vencerás. Consiste en dividir un problema grande (el arreglo completo) en subproblemas más pequeños (subarreglos), resolver esos subproblemas (ordenarlos) y luego combinar las soluciones (fusionarlos). Su complejidad es O(n log n) en el peor caso. **Pasos;**

- **Dividir** el arreglo en mitades hasta obtener subarreglos de un solo elemento.
- **Ordenar recursivamente** cada mitad.
- **Fusionar** los subarreglos ordenados en uno solo, comparando elementos de ambos.

Desventaja: requiere memoria extra proporcional a la cantidad de elementos. Accede secuencialmente a los elementos.

-Quick Sort: Algoritmo recursivo basado en el paradigma Divide y Vencerás. Consiste en dividir un problema grande en subproblemas más pequeños, reordenar los elementos según un pivote, y aplicar recursivamente el proceso a cada parte. Su complejidad promedio es O(n log n), pero en el peor caso puede ser $O(n^2)$. **Pasos:**

- Elegir un **pivote** (puede ser el primero, último, medio o aleatorio).
- **Particiona** arreglo: uno con elementos **menores o iguales** al pivote y otro con elementos **mayores**.
- Aplicar recursivamente Quick Sort sobre ambos subarreglos.
- No requiere etapa de fusión, ya que el ordenamiento se logra durante la partición.

Desventaja: Puede presentar un rendimiento pobre si el pivote no divide bien los subarreglos. Además, no es estable y puede ser sensible a la elección del pivote. Aunque no necesita memoria adicional como Merge Sort, puede consumir muchas llamadas recursivas si no se optimiza.

-Heap Sort: Usa una estructura llamada **heap**, que es como un árbol donde el número más grande siempre está arriba (en la raíz). El algoritmo tiene dos pasos:

- Primero **arma el heap** con los datos.
- Después, **saca uno por uno los valores más grandes** y los va poniendo en orden.

Siempre funciona en **O(n log n)** y no necesita memoria extra.
Es muy usado cuando se necesita un ordenamiento **seguro y eficiente**, como en sistemas embebidos.

Unidad 3: "Índices"

Un **índice** es una estructura de datos que se guarda en disco o en memoria y sirve para **acelerar el acceso** a los registros de una tabla o vista, usando una o más columnas clave. Su función es actuar como una **guía ordenada**, que permite encontrar rápidamente los datos sin tener que recorrer toda la tabla fila por fila.

El índice puede estar **integrado a la tabla** o ser una estructura **adicional**. Al crearlo, se genera un orden sobre las columnas seleccionadas, lo que mejora la velocidad de búsquedas, filtros y ordenamientos.

Sin embargo, el uso de índices también implica un **costo adicional**: ocupa más espacio y, cada vez que se insertan, actualizan o eliminan registros, el índice también debe modificarse para mantenerse actualizado.

Tipos de Acceso a Datos

- **Acceso Secuencial**: Recorre los registros en el orden físico en que fueron ingresados.
- **Acceso Secuencial Indexado:** Utiliza un índice para recorrer los datos de forma ordenada según alguna clave.
- **Acceso Directo (Random):** Accede directamente al registro por clave sin recorrer otros registros.

Métodos de indexacion

Metodo Hashing

El hashing es una técnica utilizada para almacenar y acceder a datos de manera rápida y eficiente, trabaja sobre el concepto de una tabla y una función hash (o de dispersión). Se basa en el uso de una función hash (o de dispersión) que indica dónde debe ubicarse un dato en una tabla.

-**Una función hash** es un algoritmo que transforma una clave de entrada en un índice numérico que se utiliza para ubicar el dato en una tabla. Una buena función hash debe:

- Distribuir las claves uniformemente en la tabla, y ser rápida de calcular
- Minimizar las colisiones (cuando dos claves distintas generan el mismo índice).

-Funcionamiento general

- Se crea una tabla hash, que es una estructura (normalmente un vector) con posiciones numeradas.
- Se aplica una función hash a cada clave para obtener una posición (índice).
- El dato se guarda en esa posición de la tabla.
- Para buscar un dato, se vuelve a aplicar la función hash a la clave y se accede directamente a la posición indicada.

-Colisiones: Una colisión ocurre cuando dos claves diferentes generan el mismo valor hash y, por tanto, intentan ocupar la misma posición en la tabla. Su tratamiento difiere según se trate de hashing estático o dinámico.

- En hashing estatico se aplica la función de re-hash las veces que sean necesarias hasta obtener una posición de la tabla que esté libre.
- En hashing dinámico simplemente se agrega un nodo a la lista de claves que colisionaron en esa posición.

-Técnicas de resolución de colisiones:

- **Encadenamiento:** Cada casilla de la tabla referencia una lista de los registros que colisionan en esa casilla. Se agrega al final de la lista correspondiente.
- **Direccionamiento Abierto**: Se busca otra posición dentro de la tabla. Existen varios métodos:
    - **Sondeo lineal:** Busca secuencialmente una posición vacía.
    - **Sondeo Cuadrático:** Examina posiciones a una distancia específica F(i) = i^2 del punto inicial
    - **Hashing doble:** Aplica una segunda función hash a la clave, usando el resultado como tamaño de salto

-Hashing estático: El tamaño de la tabla hash es fijo desde el inicio y no cambia, independientemente de cuántos elementos se almacenen. Es decir:

- Se reserva una cantidad fija de espacio.
- La función hash mapea claves a posiciones dentro de esa tabla de tamaño fijo.
- No se adapta automáticamente al crecimiento o decrecimiento de los datos

-Hashing into buckets: Técnica que utiliza una función hash para distribuir elementos en diferentes buckets (cubetas, compartimentos o contenedores). Cada bucket puede almacenar uno o más elementos que comparten el mismo valor hash.

- Se aplica una función hash sobre una clave.
- El resultado indica en qué bucket se almacenará el dato.
- Si varias claves tienen el mismo valor hash, van al mismo bucket, formando una lista o estructura dentro de ese compartimento.

Métodos – Arboles B

Árbol B (Btree): Tipo de árbol M-ario destinado a la creación de índices físicos para acceso a información. Su objetivo principal es minimizar las operaciones de entrada/salida al disco. Impone balance para garantizar búsqueda, inserción y eliminación en tiempo O(log n). El grado M varía basado en tamaño de claves y página de disco.

Contexto de Árboles M-arios: Necesarios para manejar grandes conjuntos de datos que no caben en memoria principal, implementando el árbol de búsqueda en almacenamiento secundario (disco). El tiempo de acceso al disco es mucho mayor que a memoria principal, por lo que hay que minimizar accesos a disco. Los discos son dispositivos orientados a bloques, transfiriendo grandes bloques eficientemente

Árboles B+

El **Árbol B+** es una **variación del árbol B**, y es el más usado para implementar **índices en bases de datos** debido a sus ventajas en el acceso secuencial y por rangos.

Características del Árbol B+:

- **Sólo las hojas** contienen punteros a los registros reales (datos).
- Los **nodos internos** solo almacenan claves para guiar la búsqueda (sin datos).
- Las hojas están **enlazadas entre sí**, permitiendo recorridos secuenciales rápidos (acceso por rango).
- También tiene complejidad **O(log n)** para búsquedas, inserciones y eliminaciones.
- Es **más eficiente** que el B-Tree para operaciones de lectura secuencial.

Ventajas del Árbol B+:

- Mejor rendimiento para **consultas por rango**.
- Mayor **densidad de claves** en nodos internos, ya que no se almacenan datos.
- Navegación más simple y uniforme en las hojas.

Comparativa;

Método	Eficiencia	Ideal Para	Soporta Rangos	Uso Común en BD
Hashing	O(1)	Búsqueda exacta	❌ No	Poco común
Árbol B	O(log n)	Búsqueda general	✅ Sí	A veces
Árbol B+	O(log n)	Búsqueda + Acceso secuencial	✅ Sí	✅ Muy común

Operaciones:

- **Búsqueda:** Similar a un árbol binario de búsqueda, pero tomando una decisión multicamino basada en el número de hijos.
- **Inserción**: Se busca el elemento comenzando en la raíz; la búsqueda sin éxito termina en un nodo hoja (punto de inserción). Si no hay espacio, se produce un **split**, que divide el nodo en dos, dejando la mitad de elementos más chicos en uno y la mitad más grandes en otro, respetando el orden.
- **Eliminación:** Se busca el elemento comenzando en la raíz; si existe, se llega a la hoja donde está y se borra. Si al eliminar un elemento el nodo queda vacío, debe eliminarse, lo que puede generar una baja potencial en los antecesores (**fusión**).
- **Fusión:** si ocurre que cuando se elimina el elemento x el nodo queda vacío, debe eliminarse el nodo, lo que puede generar una baja potencial en todos los antecesores de dicho nodo.
- **Split:** si ocurre que cuando se llega a la hoja no hay espacio para insertar el nodo se produce lo que se denomina “split” que es un proceso que divide el nodo en dos dejando la mitad de elementos en cada uno respetando el orden de menor a mayor, quedando la mitad de los elementos más chicos en un nodo y la mitad de los elementos más grandes en el otro.

Unidad 4: “Compresión de los Datos”

Compresión: Refiere al proceso de reducir el tamaño de un archivo o conjunto de datos, eliminando redundancias o representando la información de manera más eficiente. El objetivo principal es ahorrar espacio de almacenamiento y/o reducir el tiempo de transmisión de datos por redes.

Algoritmo de Huffman

Es un algoritmo para comprimir datos sin perder información. Se usa mucho con archivos de texto. Funciona dando códigos más cortos a los caracteres que más se repiten, y códigos más largos a los menos frecuentes.

-Estructuras utilizadas:

- Tabla de huffman (matriz con estado, carácter, frecuencia, código, dirección en árbol)
- Arbol binario completo: Arbol que se arma con la tabla de Huffman para obtener los códigos de caracter.
- Pila: A medida que se recorre el árbol desde una hoja hasta la raíz, se van guardando en la pila

-Proceso de Compresión:

1. **Se cuenta cuántas veces aparece cada carácter** en el texto (frecuencia), y se arma tabla de frecuencias
2. Con esa información, se construye un **árbol binario**, juntando los caracteres menos frecuentes primero.
3. En ese árbol: Ir a la izquierda es un 0 **e** Ir a la derecha es un 1
4. Para cada carácter, se arma un código binario siguiendo el camino desde la raíz hasta la hoja del árbol donde está ese carácter.
5. El archivo se comprime reemplazando los caracteres por sus códigos. También se guarda una pequeña tabla al principio con las frecuencias, para poder descomprimir luego.

-Proceso de Descompresión:

1. Primero se lee la cabecera del archivo, que dice cuántos caracteres distintos había y cuántas veces se repetía cada uno. Obteniendo las frecuencias
2. Con eso se reconstruye el árbol de Huffman original.
3. Luego se leen los bits del archivo comprimido, y con el árbol se traduce cada secuencia de 0s y 1s de vuelta a sus caracteres originales.
4. Después de identificarlo, vuelvo a la raíz y repito con los siguientes bits.

Unidad 5

Estructura de Data Base Management System (DBMS)

-BD (Base de Datos): Conjunto de datos interrelacionados que se ajustan a modelos preestablecidos, recogiendo información de interés de objetos del mundo real.

-DBMS (Sistema de Gestión de Base de Datos): Software encargado de gestionar los datos de la BD. Proporciona mecanismos de acceso eficientes para almacenar, definir y recuperar información.

Propiedades (ACID): Para ser considerado un Motor de Base de Datos, un producto debe cumplir con estas propiedades.

- **Atomicidad:** Asegura que una operación se realiza completamente o no se realiza en absoluto. Ante un fallo del sistema, no puede quedar a medias. Es imposible encontrar pasos intermedios. Si consiste en varios pasos, todos ocurren o ninguno. Característica de sistemas transaccionales.
- **Consistencia (Integridad):** Asegura que solo se empieza aquello que se puede terminar. Solo se ejecutan operaciones que no rompen las reglas de integridad de la base de datos. Cualquier transacción lleva la base de datos de un estado válido a otro válido. Garantiza que los datos sean exactos, consistentes, intactos y no se deformen.
- **Aislamiento:** Asegura que una operación no afecta a otras. La realización de dos transacciones sobre la misma información son independientes y no generan errores. Define cómo y cuándo los cambios de una operación son visibles para otras concurrentes. El nivel de aislamiento es esencial al seleccionar un DBMS.
- **Durabilidad (Persistencia):** Asegura que una vez realizada la operación, esta persistirá y no se podrá deshacer aunque falle el sistema. Los datos sobreviven. Persistencia es la acción de preservar la información de un objeto de forma permanente (guardado) y poder recuperarla (leerla).

Teoria de objetos de base de datos

Vistas(view): una vista es una tabla virtual cuyo contenido está definido por una consulta. Los datos de esta tabla proceden de las tablas o vistas a las que se referencia en la consulta y pueden pertenecer a otra base de datos. De esta forma, es una presentación adaptada de los datos contenidos en una o más tablas, o en otras vistas.

- No ocupa espacio de almacenamiento para datos (salvo vistas indexadas).
- Se puede consultar como si fuera una tabla.
- Puede combinar múltiples tablas en una sola vista.
- Las vistas son una excelente herramienta para implementar seguridad en una base de datos;
    - Restricción de acceso a columnas y filas => Muestra campos necesarios
    - Evitar acceso directo a tablas
    - Encapsulamiento de lógica compleja

Snapshot: Un snapshot es una captura del estado de los datos en un momento determinado.
Es una copia lógica, no física, que permite ver o restaurar la información tal como estaba en ese instante.

- Es **de solo lectura** (no modificable)
- No duplica toda la base, solo **registra los cambios** posteriores (eficiente)
- Puede ser usado para **comparar estados** o revertir operaciones

- Se puede aplicar a bases de datos, discos, archivos o máquinas virtuales

Diferencias entre una vista y un snapshot: al igual que una view, un snapshot tiene un select a una o más tablas pero cada vez que se invoque a este snapshot en un from en vez de mostrar los datos al momento va a mostrar los datos a la última “foto” que se les haya sacado. El snapshot en sí tiene una sentencia de refresh que le indica cada cuanto tiene que volver a ejecutarse.

TABLA TEMPORAL

Definición: es una tabla cuyos datos son de existencia temporal y tiene las siguientes características**:**

- Solo es visible para la sesión actual. No puede ser vista o utilizada por procesos o consultas fuera de la sesión en la que esta se declara.
- No son registradas en las tablas de diccionario de datos.
- No es posible alterarlas, pero sí eliminarlas y crear los índices temporales que necesite.

Conviene usarlas, por ejemplo, como almacenamiento intermedio en consultas muy grandes ya que pueden servir para guardar resultados intermedios basados en consultas de menor tamaño.
También suelen usarse en ciclos, para almacenar elementos que el ciclo necesite leer. Proporciona un medio rápido y eficiente para hacerlo.

Unidad 6: Inteligencia de Negocios y Tecnologías OLAP

La Inteligencia de Negocios (Business Intelligence) es el proceso mediante el cual se transforman datos en información útil y esta, a su vez, en conocimiento, con el objetivo de mejorar la toma de decisiones en una organización.

OLAP es una tecnología usada para analizar datos complejos desde distintas perspectivas, ideal para la toma de decisiones estratégicas. Trabaja con bases de datos multidimensionales, organizadas por dimensiones y medidas.

Está pensado para consultas rápidas y complejas, no para operaciones del día a día. Por eso, los datos están desnormalizados y se cargan desde otras fuentes mediante procesos ETL (extracción, transformación y carga).

Los datos en OLAP son históricos, persistentes y no cambian con frecuencia. Se duplican para mejorar el rendimiento, evitar afectar al sistema transaccional (OLTP) y permitir análisis consistentes.

Por otro lado, **OLTP** es el modelo usado en operaciones diarias como ventas o pagos. Es rápido, seguro, usa datos normalizados, se actualiza constantemente y maneja información detallada y volátil.

OLAP es para análisis de alto nivel, OLTP para actualizaciones transaccionales a nivel de detalle.

Bases de Datos Multidimensionales (BDM)

Las **dimensiones** determinan la estructura de la información almacenada y definen caminos de consolidación. La información se presenta como variables caracterizadas por una o más dimensiones. La información puede analizarse dentro del **cubo** formado por la intersección de las dimensiones.

Un **cubo OLAP** es una estructura multidimensional que permite analizar grandes volúmenes de datos desde diferentes perspectivas (dimensiones). Cada **cubo** contiene:

- **Medidas** (datos cuantitativos, ej: ventas, ingresos)
- **Dimensiones** (categorías para analizar, ej: tiempo, producto, región)

Hipercubo: La información se guarda implícitamente en un gran cubo único, presentando los datos al usuario en un formato de hipercubo. Todos los datos aparecen como una sencilla estructura multidimensional.

Multicubo: La información se almacena dividiendo los datos en grupos más pequeños y densos para proporcionar un análisis más amplio y detallado.

- Permite combinar y cruzar información de distintos cubos.
- Facilita análisis complejos que involucran diferentes áreas o niveles de detalle.
- Ayuda a mejorar el rendimiento al especializar cubos para diferentes temas y luego combinarlos.

Diseño de Multicubo

1. **Identificar los temas o áreas de análisis** que se desean modelar (ejemplo: ventas, inventarios, finanzas).
2. **Diseñar cubos individuales** para cada tema, definiendo sus medidas y dimensiones.
3. **Definir relaciones entre cubos**, que pueden ser:
    a. Cubos con dimensiones comunes (ej: tiempo)
    b. Cubos dependientes, donde uno incluye dimensiones o medidas del otro
4. **Integrar los cubos mediante enlaces o esquemas de navegación**, para permitir análisis combinados.
5. **Optimizar el almacenamiento y consulta**, evitando redundancia y maximizando la eficiencia.

Encriptación

la **Encriptación** es el proceso que permite modificar el contenido de un archivo para que el mismo no pueda ser visible en un formato tradicional, manteniendo el contenido establecido pero oculto. Su objetivo principal es **ocultar la información** contenida en el archivo para que no sea legible. Es importante que, al encriptar, el archivo sea modificado sin cambiar su tamaño ni el espacio ocupado.

Procesos de encriptación:

Desplazamiento: Se basa en el desplazamiento de los caracteres según algún patrón. Un método de desplazamiento se realiza por bits, similar a una "sopa de letras", modificando totalmente el contenido.

Reemplazo: Consiste en reemplazar determinados caracteres en función de algún patrón. Este reemplazo puede ser fijo o variable, con o sin intervención del usuario.

> **Reemplazo Fijo:** Se toma un valor por el cual se reemplazan determinados caracteres según un patrón (ej. reemplazar posiciones pares por un valor ASCII preestablecido).
>
> **Reemplazo Variable:** El archivo se encripta con una clave dispuesta por el usuario, y dicha clave se copia al contenido del archivo o valor a encriptar.

Mixto: Se aplican ambos procesos (reemplazo y desplazamiento) en cualquier orden para proporcionar mayor seguridad a la encriptación.

Integridad

La **integridad** es la capacidad de asegurar que un archivo no ha sido modificado o alterado sin autorización desde su última versión válida. Esto garantiza que los datos sean **correctos, completos y coherentes**, evitando errores, duplicaciones o cambios no deseados con el tiempo.

-Control de Integridad: Necesidad de verificar si un archivo modificado es igual al original.

Procedimiento: Aunque la única forma segura es comparar byte a byte, existen medios tecnológicos para validar la veracidad sin el original. Para considerar que dos archivos son iguales, se deben cumplir tres condiciones:

- Tener el **mismo tamaño** (misma cantidad de caracteres),
- El **mismo contenido**,
- Y los caracteres deben estar en la **misma posición**

Checksum: Consiste en crear un **polinomio** con los caracteres del archivo. Cada carácter se convierte en bits, y estos activan ciertas potencias del polinomio. Al final se obtiene un número (el checksum) que se adjunta al archivo. Si alguien lo modifica, al volver a calcular el polinomio ese número no va a coincidir, revelando que hubo cambios.

CRC (Control de Redundancia Cíclica): Es un tipo de checksum más avanzado, usado en comunicaciones y almacenamiento. Calcula un código (de 32 a 128 bits, por ejemplo) a partir del contenido del archivo. Al recibir el archivo, se vuelve a calcular y se compara con el código original. Si no coinciden, es porque hubo una alteración.

-BD Constraints, es un objeto de base de datos que se asocia a la tabla, y realiza las verificaciones de integridad. Los tipos de Constraints son;

- PRIMARY KEY: que no haya pk repetida y que no tenga nulos.
- FORENIGN KEY: uno o mas campos que coincidan a cantidad y estructura respecto a la PK.
- UNIQUE: Garantiza unicidad de campos, pero no saldrá a validar en otras tablas.
- NOT NULL: Verifica el campo sea no nulo.
- CHECK: Expresión que debe cumplir vof, no podemos chequear contra datos del sistema.

- DEFAULT: Cuando se inserta un registro en la tabla, si no se define un valor se asigna un default.

Valores nulos y desconocidos: Los valores nulos representan datos faltantes. Ejemplo: un estudiante nuevo puede no tener registrada su localidad de origen inicialmente. SQL utiliza NULL para representar estos valores.

-Transacción, es una **unidad lógica de trabajo** que consiste en una o más operaciones SQL (como INSERT, UPDATE, DELETE) que se ejecutan **como un solo bloque**.
El objetivo de una transacción es que los **datos permanezcan consistentes**, incluso ante errores, caídas o accesos simultáneos.

Cumple con las siguientes propiedades ACID:

- **ATOMICIDAD:** Todas las operaciones se realizan completamente o no se realiza ninguna..
- **CONSISTENCIA (Integridad)**: Despues de ejecutar una transacción, la base de datos queda en un estado consistente porque los datos pasan de un estado válido a otro estado válido.
- **ISOLATION (AISLAMIENTO):** Las transacciones se ejecutan como si fueran solas (evitan interferencias).
- **DURABILIDAD:** Una vez que se confirma el dato, queda perpetuo en la base de datos.

Ciclo de una transacción

1. **BEGIN TRANSACTION** – Comienza la transacción
2. Se ejecutan operaciones SQL (INSERT, UPDATE, DELETE)
3. **COMMIT** – Confirma los cambios permanentemente
4. **ROLLBACK** – Revierte todos los cambios si hay error