Tema 1

1. Introducción al tema
Objetivo: Conocer los retos y criterios básicos para diseñar y evaluar un sistema de gestión de datos eficiente a gran escala.

Bloque 1 – Eficiencia en grandes cantidades de datos

- **Por qué importa**
  - A medida que crece el volumen de información, las operaciones de lectura/escritura pueden volverse muy costosas en tiempo y recursos.
  - Un diseño apropiado minimiza el número de accesos al disco y aprovecha mejor la memoria intermedia (buffers).
- **Problemas a resolver**
  1. **Almacenamiento**: ¿Cómo organizar físicamente millones de registros sobre dispositivos masivos?
  2. **Acceso rápido**: ¿Cómo localizar y recuperar un registro concreto de manera inmediata?
  3. **Arquitecturas de datos**: ¿Qué niveles de abstracción se usan para gestionar la complejidad? □cite□turn0file0□

Bloque 2 – Forma de almacenamiento de los datos

1. **Niveles de abstracción**
   - **Nivel externo**: visión de usuario (vistas personalizadas).
   - **Nivel conceptual**: diseño lógico global (modelo entidad-relación, tablas).
   - **Nivel físico**: organización efectiva en bloques y ficheros sobre disco.
2. **Impacto en rendimiento**
   - La disposición de registros en bloques condiciona los tiempos de lectura secuencial y aleatorio.
   - Un buen diseño físico reduce fragmentación y desperdicio de espacio. □cite□turn0file0□

Bloque 3 – Forma de acceso rápido a los datos

- **Accesos secuenciales vs. directos**
  - *Secuencial*: ideal para lecturas completas, pero lento para encontrar un registro concreto.
  - *Directo*: acceso por clave mediante índices o funciones de hash, minimiza desplazamientos de cabezal.
- **Métricas clave**
  - Tiempo para encontrar un registro arbitrario (T).
  - Tiempo para encontrar por clave (TF).
  - Tiempo para leer siguiente registro (TN).
  - Estas métricas guían la elección de estructuras de índice y organización de bloques. □cite□turn0file0□

Bloque 4 – Arquitecturas para relacionar los datos

1. **Modelo cliente-servidor**
    - Separación clara de responsabilidades: servidor gestiona almacenamiento y concurrencia; cliente presenta/solicita datos.
2. **Capas del SGBD**
    - *Procesamiento de consultas*: optimizador y planificador.
    - *Gestión de almacenamiento*: motor interno que controla bloques, buffers y ficheros.
    - *Control de concurrencia*: bloqueo y transacciones para garantizar integridad.
3. **Ventajas**
    - Escalabilidad horizontal y vertical según la carga de trabajo.
    - Mantenimiento y actualización modularizada. □cite□turn0file0□

Bloque 5 – Medidas para evaluar un sistema de archivos
Objetivo: cuantificar los costes de espacio y tiempo asociados a distintas organizaciones físicas, para poder comparar sistemas de ficheros y modos de acceso. □cite□turn0file0□

---

1. **Parámetros evaluables**

    - **R**: memoria necesaria para almacenar un registro (bytes).
    - **T**: tiempo de acceso al disco para localizar un registro arbitrario.
    - **TF**: tiempo de localizar un registro dado su valor de clave.
    - **TW**: tiempo de escribir un registro cuando ya conocemos su posición física.
    - **TN**: tiempo de encontrar el siguiente registro a uno dado.
    - **TI**: tiempo de insertar un nuevo registro (ampliar fichero).
    - **TU**: tiempo de actualizar un registro existente.
    - **TX**: tiempo de leer todo el archivo (recorrido completo).
    - **TY**: tiempo de reorganizar el fichero (p.ej. tras muchas eliminaciones/inserciones). □cite□turn0file0□

2. **Operaciones fundamentales**

    - **Recuperar un registro por clave**
      – Coste: TF (depende de si hay o no índice, y de su tipo).
    - **Obtener el siguiente registro**
      – Coste: TN (lectura secuencial dentro del bloque o mediante punteros).
    - **Insertar un registro**
      – Coste: TI (en un ASF es simplemente TW; en formatos ordenados puede implicar TF + desplazamientos).
    - **Actualizar un registro**
      – Si no cambia tamaño: TU = TF + TW.
      – Si cambia tamaño: TU = TF + TW + TI (hay que moverlo o reinsertarlo).
    - **Leer todo el fichero**
      – Coste TX ≈ Nº de bloques × T (acceso secuencial).
    - **Reorganizar el fichero**
      – Coste TY incluye lectura de todos los registros más reescritura (depende de nº eliminados y añadidos). □cite□turn0file0□

---

3. **Por qué importa**
   - Con estos parámetros puedes **comparar** dos métodos (p.ej. ASF vs. ASL) y elegir el más rápido para tu patrón de acceso.
   - Permiten predecir **rendimiento** en inserciones masivas, búsquedas por rango o barridos exhaustivos.

> **Tip de estudio:** memoriza primero la lista de parámetros y operaciones (punto 1 y 2). Luego, cuando veas un método concreto (ASF, ASL, ASI, AAD), podrás sustituir TF, TI, etc., por las fórmulas correspondientes.

Bloque 6 – Archivo Secuencial Físico (ASF)
Objetivo: entender cómo funciona el método más simple de almacenamiento y cómo estimar sus costes en tiempo y espacio. □cite□turn0file0□

---

1. **Definición y características**

   1. Los registros se almacenan tal cual, uno tras otro, sin orden lógico interno.
   2. Cada campo es un par `(Attr_id, valor)`; se usan dos separadores:
      - Entre identificador y valor (e.g. `=`).
      - Entre campos sucesivos (e.g. `;`).
   3. Útil cuando:
      - No se requiere acceso por rango o clave con frecuencia.
      - Predomina la inserción rápida y la lectura exhaustiva del fichero.

2. **Cálculo del tamaño de registro (R)**

   - Fórmula:

     ```
     R = a' × (A + V + 2)
     ```

     donde:
     - a' = número medio de atributos por registro
     - A = longitud media (bytes) de cada identificador de atributo
     - V = longitud media (bytes) de cada valor de atributo
     - 2 = dos separadores por atributo (`=` y `;`) □cite□turn0file0□

3. **Operaciones y sus costes**

   - Sea n = nº de registros ya existentes, T = tiempo de acceso a bloque.

   1. **Recuperar un registro por clave (TF)**

      ```
      TF = (n + 1) / 2 × T   ≈ n/2 × T
      ```

      (búsqueda secuencial hasta hallar la clave)

2. **Recuperar siguiente registro (TN)**

```
TN = TF
```

3. **Insertar registro (TI)**

```
TI = TW
```

(solo escribir al final; TW = tiempo de escritura conocido)

4. **Actualizar registro (TA)**
   - Si **no** cambia el tamaño:

```
TA = TF + TW
```

   - Si **sí** cambia el tamaño (hay que reubicar):

```
TA = TF + TW + TI
```

5. **Leer todo el fichero (TX)**

```
TX = n × T
```

(barrido completo, bloque a bloque)

6. **Reorganizar el fichero (TY)**

```
TY = (n + O) × T  +  (n + O − d) × TW
```

donde:
- O = nº de registros añadidos desde la última reorganización
- d = nº de registros marcados para borrar □cite□turn0file0□

4. **Ventajas y desventajas**

   - **Ventajas**
     - Inserción inmediata y sencilla (`TI = TW`).
     - Lectura secuencial muy eficiente (`TX` lineal en n).
   - **Desventajas**
     - Búsqueda por clave muy lenta (O(n)).
     - No soporta accesos directos ni rangos; obliga siempre a escaneo total.

5. **Conceptos clave para el examen**
    - Definición de ASF y cuándo usarlo.
    - Fórmula de tamaño de registro R.
    - Interpretar y aplicar las fórmulas de TF, TI, TX y TY.
    - Significado de los parámetros O y d en reorganización.

Bloque 7 – Archivo Secuencial Lógico (ASL)
Objetivo: comprender cómo mantener un fichero ordenado por clave y evaluar su rendimiento cuando hay inserciones y una "zona de desbordamiento". □cite□turn0file0□

---

1. **Definición y características**
    - Los registros están **ordenados** físicamente según el valor de una clave (uno o varios campos).
    - Todos los registros tienen **longitud fija** y estructura uniforme; el formato se almacena en la cabecera del fichero.
    - Para no reescribir el fichero completo en cada inserción, se usa una **zona de desbordamiento** (así como un ASF): los nuevos registros van ahí sin orden.
    - Cuando la zona de desbordamiento crece demasiado, es necesario **reconstruir** (reordenar) todo el fichero. □cite□turn0file0□

2. **Tamaño de registro (R)**
    - Al ser fijos,

      ```
      R = ∑ (longitud de cada campo i)
      ```

    - La cabecera ocupa espacio extra, pero suele considerarse constante y pequeño. □cite□turn0file0□

3. **Operaciones y costes**
   Sea n = nº de registros en el fichero maestro, O = nº en zona de desbordamiento, T = tiempo de lectura de bloque, TW = tiempo de escritura.

Operación	**Coste aproximado**
Recuperar registro (valor NO clave)	TF = (n/2)·T (escaneo en maestro)
+ con overflow: TF = (n/2 + O/2)·T	
Recuperar registro (clave)	TF ≈ $\log_2(n)$·T (búsqueda binaria en maestro)
Siguiente registro	TN = T (si está en maestro)
o TN = T + O·T (si en overflow)	
Insertar registro	• En overflow: TI = TW
• En maestro (sin overflow):	TI = TF + (n/2)·T + (n/2)·TW [+ TW]

Operación	Coste aproximado
Actualizar registro	• Si clave no cambia: TU = TF + TW
• Si cambia clave: TU = TF + TW + TI	
Leer todo el fichero	• Sólo maestro: TX = n·T
• Con overflow: TX = TC(O) + (n+O)·T	
Reorganizar fichero	TY = TC(O) + (n+O)·T + (n+O–d)·TW
	(d = nº registros borrados)

4. **Ventajas y desventajas**
    - **Ventajas**
        - Las búsquedas por clave son mucho más rápidas que en ASF (O(log n) vs O(n)).
        - Se mantiene orden sin reescribir todo en cada inserción (hasta que O crece).
    - **Desventajas**
        - Abrir hueco para nuevas inserciones en el maestro es costoso; dependen de TF y de n.
        - La zona de desbordamiento degrada la reconstrucción y la lectura secuencial.
        - Hay sobrecarga al reorganizar (TY puede ser alto si O o d son grandes).

5. **Puntos clave para el examen**
    - Definición de ASL y la función de la zona de desbordamiento.
    - Fórmula de R para registros fijos.
    - Diferencia entre TF en búsqueda de clave ($\log_2$ n·T) y no clave (n/2·T).
    - Coste de inserción en overflow (TW) vs. en maestro (TF + desplazamientos).
    - Entender cuándo y por qué se reorganiza el fichero (fórmula TY).

Bloque 8 – Archivo Secuencial Indexado (ASI)
Objetivo: acelerar el acceso por clave mediante un índice asociado al fichero de datos. □cite□turn0file0□

---

1. **Definición y estructura**
    - Un ASI consta de dos ficheros:
        1. **Fichero de datos**: similar a un ASL (ordenado y/o con zona de desbordamiento).
        2. **Fichero de índice**: estructura ASL de registros de longitud fija que contienen
            - Valor de clave (Vk)
            - Puntero a **registro** (índice denso) o a **bloque** (índice no denso) en el fichero de datos. □cite□turn0file0□
    - **Índice denso**: una entrada por cada registro de datos.
    - **Índice no denso**: una entrada por cada bloque de datos (menor espacio, pero busca bloque + escaneo).

2. **Tamaño de un registro de índice**
    - Denso:

```
R_idx = ∑(longitud campos de índice) = (V_k + P)
```

donde Vk = longitud del valor de clave, P = tamaño del puntero.

- No denso: idéntico, pero el nº de entradas es n/Bfr (registros por bloque). □cite□turn0file0□

3. **Operaciones y costes**
   Sea n = nº registros, Bfr = factor de bloqueo de datos, T = tiempo de lectura bloque de datos, TFi = tiempo de lectura bloque de índice, TW = tiempo de escritura.

Operación	Índice Denso	Índice No Denso
Recuperar por clave	TF = $\log_2(n)$·TFi + T	TF = $\log_2(n/Bfr)$·TFi + T + T·Bfr/2
Recuperar siguiente registro	TN = TFi + T = 2·T	TN = TFi + T (si en mismo bloque)
Insertar registro	• Maestro sin overflow:	

TI = (TF_datos + TF_idx) + (n/2)·TW + (n/2)·TW
⇒ TI ≈ 2·(TF + n/2·TW)
• Con overflow: TI = TW + (TF_idx) | • Similar, pero TF en índice sobre n/Bfr
y escritura en bloque de overflow | | **Actualizar registro** | • Sin cambio de clave: TU = TF + TW
• Con cambio de clave: TU = 2·(TF + TW) + TI_datos + TI_idx | • Análogo, usando TF y TI del índice no denso | | **Leer todo el fichero** | TX = n·T (datos) + n·TFi (índice si se recorre) ≈ 2·n·T | TX ≈ n·T + (n/Bfr)·TFi | | **Reorganizar fichero** | TY = TC(O) + (n+O)·T + (n+O–d)·TW + (n+O–d)·TW | Análogo adaptado a entradas por bloque | □cite□turn0file0□

4. **Ventajas y desventajas**

   - **Ventajas**
     - Búsqueda por clave muy rápida (O(log n)).
     - Mantiene datos ordenados sin escanear todo.
   - **Desventajas**
     - Sobrecarga de espacio adicional para índice (índice denso puede ser tan grande como datos).
     - Mantenimiento de índice en inserciones/actualizaciones (overhead en escritura).
     - Para índices no densos, tras localizar bloque queda escaneo interno.

5. **Puntos clave para el examen**

   - Distinguir **denso** vs **no denso**: entradas por registro vs por bloque.
   - Fórmulas básicas de TF: $\log_2(n)\cdot TFi + T$ (denso) y $\log_2(n/Bfr)\cdot TFi + T + (Bfr/2)\cdot T$ (no denso).
   - Coste extra de mantener el fichero de índice en inserciones y actualizaciones.
   - Reconocer cuándo conviene un ASI (acceso frecuente por clave) y elegir tipo de índice según memoria disponible.

Bloque 9 – Índice multinivel (ASI multinivel)
Objetivo: reducir el número de accesos a disco cuando el índice principal es muy grande. □cite□turn0file0□

- **Motivación**
  - Un índice denso sobre millones de registros puede ocupar tanto espacio que ni siquiera cabe en memoria.
  - Solución: indexar el índice, formando varios niveles (árbol de índices).
- **Estructura**
  1. **Nivel 1 (raíz):** contiene una entrada por cada bloque del fichero de datos.
  2. **Nivel 2:** índice del índice; una entrada por cada bloque de nivel 1.
  3. …
  4. **Nivel m (hojas):** apunta a los bloques de datos donde están los registros.
- **Cálculo aproximado de entradas y bloques**
  - Sea y = nº de entradas que caben en un bloque de índice = ⌊(B–C)/(V+P)⌋.
  - i_1 = n/Bfr (entradas de nivel 1).
  - b_1 = ceil(i_1 / y) (bloques de nivel 1).
  - i_2 = b_1, b_2 = ceil(i_2 / y), … hasta que bj = 1.
  - m = número de niveles (altura del árbol).
- **Coste de búsqueda por clave (TF)**

  ```
  TF ≈ m·TFi + T
  ```

  donde TFi = tiempo de leer un bloque de índice, T = leer bloque de datos.
- **Ventajas y desventajas**
  - Ventaja: m suele ser muy pequeño (2–3), así que búsquedas O(m) ≪ O($\log_2$ n).
  - Desventaja: más complejidad al insertar/actualizar (hay que mantener varios niveles).

Bloque 10 – Archivo de Acceso Directo (AAD)
Objetivo: permitir acceso “directo” a registros mediante funciones de hash con coste promedio O(1).
□cite□turn0file0□

- **Idea básica**
  - Se elige m celdas (espacios) en el fichero para n registros, con m ≥ n.
  - Una función de hash convierte cada clave en un valor entre 0 y m–1.
- **Colisiones y huecos**
  1. **Colisiones** (dos claves mapean la misma celda):
     - **Direccionamiento cerrado** (tabla única):
       - *Búsqueda lineal*: probar la siguiente celda libre.
       - *Re-hashing*: aplicar segunda función hash si colisión.
     - **Direccionamiento abierto** (desbordamiento):
       - *Listas enlazadas*: colisiones en fichero de overflow.

        - *Bloques de desbordamiento*: agrupar varios registros por bloque.
    2. **Huecos** (celdas vacías):
        - Se dejan intencionadamente para reducir colisiones; factor de carga α = n/m.

- **Hashing dinámico**
    - Extensible o lineal: la tabla crece o se reorganiza automáticamente para mantener α bajo control.
- **Costes promedio**
    - **TF** ≈ C + α·T (C = coste de cómputo hash, α = nº medio de colisiones por búsqueda).
    - **TI** = C + 2·TW (calcular hash y escribir en overflow si hay colisión).
    - **TX** ≈ (m+c)·T (barrido completo de todas las celdas más overflow).
- **Ventajas y desventajas**
    - Ventaja: acceso muy rápido en promedio, independiente de n.
    - Desventaja: rendimiento cae si α crece; reorganización y espacio extra pueden ser costosos.

Bloque 11 – Evaluación del sistema
Objetivo: saber cómo estimar cargas, costes y beneficios para elegir el diseño físico óptimo. □cite□turn0file0□

1. **Estimar demanda de almacenamiento**
    - Basada en: nº de registros, nº total de atributos, longitudes medias de campos e identificadores.
    - Calcular espacio necesario + margen para crecimientos futuros (densidad D).
2. **Estimar carga de recuperación**
    - Contar nº de búsquedas, inserciones, actualizaciones en un periodo (transacciones/hora).
    - Asignar cada operación a un método físico (ASF, ASL, ASI, AAD) y usar TF, TI, TU para estimar tiempo total.
3. **Estimar costes de actualización**
    - Frecuencia de inserción, borrado y modificación; calibrar TY (reorganización) y TU.
4. **Análisis coste-beneficio**
    - **Beneficios:** tiempo medio de respuesta, productividad del personal (menor latencia, menor coste de uso).
    - **Costes:** hardware (discos, controladoras), desarrollo y mantenimiento (personal, migraciones).
    - Elegir la combinación de diseño físico y recursos que minimice coste total sin degradar el servicio.
5. **Proceso iterativo**
    1. Estimar demanda.
    2. Estimar recursos disponibles.
    3. Comparar alternativas.
    4. Ajustar (o cambiar equipamiento / diseño).
    5. Repetir hasta converger en una solución satisfactoria.

Tema 2

Tema 3

Bloque 1 – Diccionario de datos o catálogo
Objetivo: conocer dónde y cómo almacena el SGBDR la metadata de la base (esquemas, objetos, permisos, etc.) □cite□turn1file0□

1. **Qué es**
    - Conjunto de tablas base y vistas que describen los objetos del SGBD: tablas, vistas, índices, usuarios, privilegios, auditoría, tablespaces, datafiles…
2. **Composición**
    - **Tablas**: atributos, tipos de datos, restricciones, propietarios.
    - **Vistas**: nombre, consulta asociada, propietario.
    - **Índices**: nombre, tabla asociada, columnas indexadas, tipo.
    - **Extras**: roles, auditoría, información de almacenamiento (tablespaces, datafiles).
3. **Organización**
    - Se almacena en el **tablespace SYSTEM**.
    - El SGBD crea tablas base (solo accesibles a SYS) y vistas (USER_, *ALL_*, DBA_*):
        - **USER**_*: objetos del usuario actual.
        - **ALL**_*: objetos accesibles al usuario.
        - **DBA**_*: acceso bruto a las tablas de catálogo (solo SYS o roles con privilegio).
4. **Utilidad práctica**
    - **Organizar** definiciones de objetos y relaciones.
    - **Mantener** coherencia e integridad (p.ej. restricciones registradas).
    - **Consultar** metadatos para optimización, auditoría y mantenimiento.
5. **Ejemplo de entrada en catálogo**

```
CREATE TABLE CARD (
  CARDID    VARCHAR2(20) PRIMARY KEY,
  CARDNAME  VARCHAR2(30) NOT NULL,
  ACCOUNTNO VARCHAR2(20) NOT NULL
    REFERENCES ACCOUNT,
  EXPDATE   DATE       NOT NULL,
  DAILYLIMIT NUMBER(4) NOT NULL CHECK (DAILYLIMIT>=0),
  LASTLIMIT  NUMBER(6,2) NOT NULL
    CHECK (LASTLIMIT>=0 AND LASTLIMIT<=DAILYLIMIT)
);
```

Cada línea y restricción queda reflejada en el diccionario de datos. □cite□turn1file0□

Bloque 2 – Estructura interna de un SGBD relacional
Objetivo: entender cómo Oracle gestiona físicamente el espacio en disco para tablas, índices y otros objetos. □cite□turn1file0□

1. **Jerarquía de objetos físicos**

```
Tablespace → Datafile(s) → Segmentos → Extensiones → Bloques
```

2. **Tablespaces y datafiles**
   - Un **tablespace** agrupa uno o más **datafiles** (archivos ÓS).
   - Se crea con `CREATE TABLESPACE … DATAFILE 'ruta' SIZE X;` y puede autoextenderse o añadirse datafiles con `ALTER TABLESPACE` / `ALTER DATABASE`.

3. **Segmentos y extensiones**
   - Al crear una tabla o índice se asigna un **segmento**: colección de **extensiones**.
   - Cada extensión: conjunto contiguo de bloques; al finalizar, se encadena otra extensión en el mismo segmento.
   - Tipos de segmentos:
     - **Datos** (tablas)
     - **Índice** (índices)
     - **Temporales** (ordenaciones, agrupamientos)
     - **Rollback** (antes de cambios en UPDATE).

4. **Bloques internos de Oracle**
   - Cada bloque contiene:
     - **Cabecera** (dirección, tipo de segmento).
     - **Directorios** (de tablas y de filas que ocupan el bloque).
     - **Zona de datos** (tuplas empacadas, posibles fragmentos si la fila no cabe entera).
     - **Espacio libre** para futuros inserts o poста.

5. **Alternativas de organización**
   - **Un fichero por relación**: cada tabla/índice en su propio datafile.
   - **Varias relaciones en el mismo fichero**: ahorra archivos ÓS, pero puede fragmentar el espacio y complicar backups.
   - Elección basada en facilidad de mantenimiento versus rendimiento de I/O. □cite□turn1file0□

Bloque 3 – Estructura lógica de un SGBD relacional
Objetivo: conocer los objetos lógicos (esquemas, tablas, vistas, índices, clústers) que modelan los datos y definen el acceso. □cite□turn1file0□

1. **Esquema**

- Conjunto de objetos que pertenecen a un usuario: tablas, vistas, índices, procedimientos, paquetes, disparadores…

2. **Tablas**

   - Definidas por columnas (atributos) y filas (tuplas).
   - Cada tabla tiene un **segmento** con un **rowid** único por fila.
   - Si una fila excede el tamaño de bloque, se divide y enlaza en otro bloque.
   - Sentencias clave:

```
CREATE TABLE … TABLESPACE users
  STORAGE (INITIAL 100K NEXT 100K MAXEXTENTS 10);
DROP TABLE …;
```

3. **Vistas**

   - Consulta almacenada sobre una o varias tablas/vistas.
   - No materializan datos (salvo vistas materializadas, no tratadas aquí).
   - Restricciones para que sean actualizables: sin agregaciones, DISTINCT, uniones ni conjuntos, y con todas las NOT NULL.
   - Uso: seguridad, abstracción, simplificación de consultas, aislamiento de cambios.

4. **Índices**

   - Estructuras (normalmente B* Trees) que agilizan búsquedas por columna(s).
   - Coste: ocupan espacio y enlentecen INSERT/UPDATE/DELETE.
   - Buenas prácticas: crear después de cargar datos; evitar índices redundantes o sobre columnas muy volátiles.
   - Se consultan en DBA_INDEXES y DBA_IND_COLUMNS.

5. **Clústers**

   - Agrupan físicamente tablas que comparten clave de unión para optimizar “joins” frecuentes.
   - Se crea con `CREATE CLUSTER` y luego las tablas con `CLUSTER nombre (columnas)`, finalizando con `CREATE INDEX … ON CLUSTER`.
   - Desventaja: menos eficiente para accesos individuales y mantenimiento complejo.
