UNIVERSIDAD DE GRANADA

DECSAI

# Administración de Bases de Datos

## Grado en Ingeniería Informática

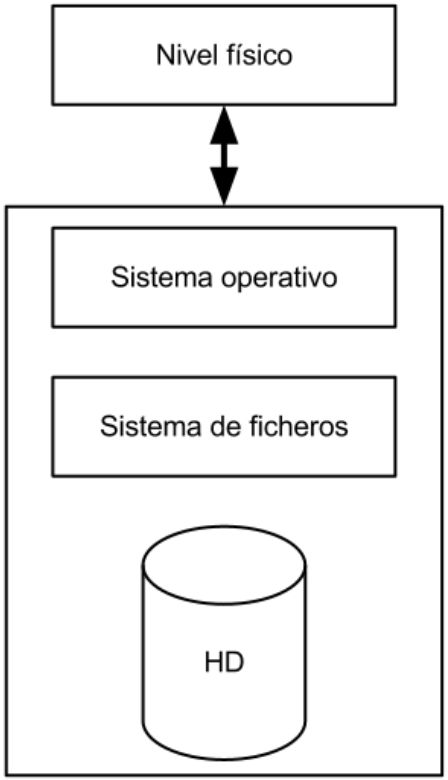
## Organización de los datos en un SGBD Relacional (SGBDR)


I. J. Blanco, A. G. López Herrera

**Departamento de Ciencias de la Computación e Inteligencia Artificial**

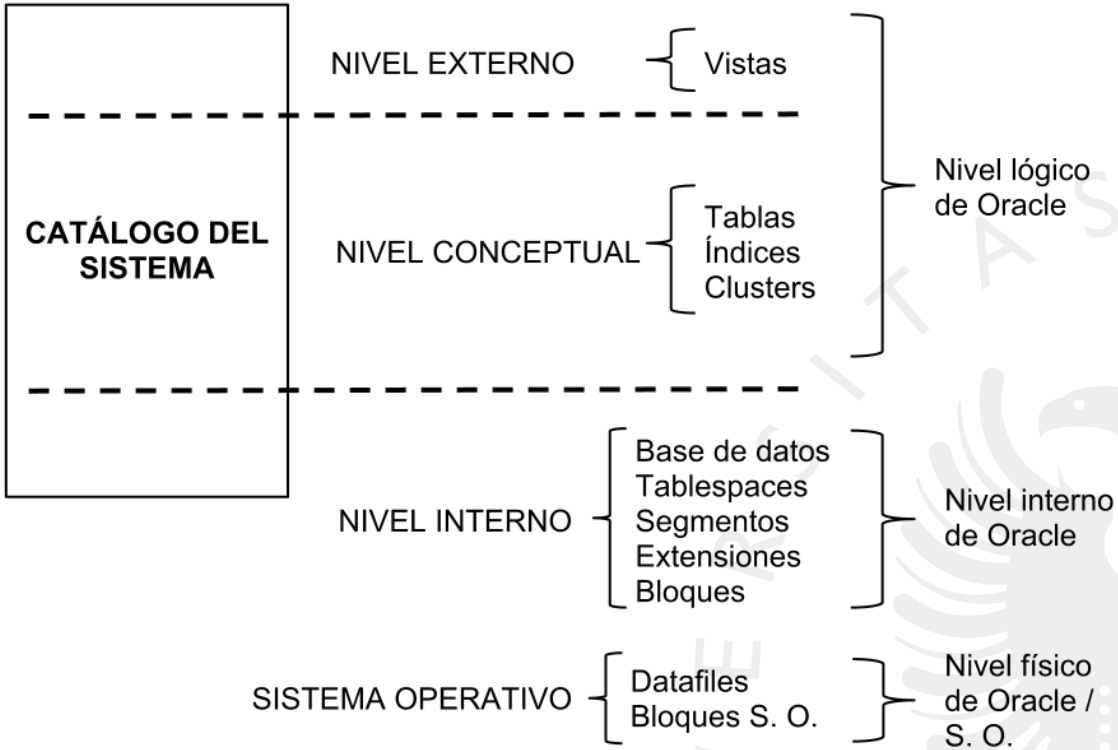
**http://decsai.ugr.es**

- El diccionario de datos o catálogo
- Estructura interna de un SGBD relacional
- Estructura lógica de un SGBD relacional




- Alternativas:
  - Un fichero por relación
  - Varias relaciones en el mismo fichero

- Imagina el problema de las cuentas y los movimientos. Encontrar todas las cuentas de un cliente con sus correspondientes movimientos
- Buscar las cuentas
- Buscar los clientes
- Transferir ambos a bloques de memoria


CATÁLOGO DEL SISTEMA
NIVEL EXTERNO
Vistas
NIVEL CONCEPTUAL
Tablas
Índices
Clusters
Nivel lógico de Oracle
NIVEL INTERNO
Base de datos
Tablespaces
Segmentos
Extensiones
Bloques
Nivel interno de Oracle
SISTEMA OPERATIVO
Datafiles
Bloques S. O.
Nivel físico de Oracle / S. O.

- Conjunto de estructuras para almacenar información sobre los datos.

- Objetos:
  - Tablas (atributos, tipos de datos, restricciones, propietario, ...)
  - Vistas (nombre, consulta asociada, propietario, ...)
  - Índices (nombre, tabla, atributos, tipo, propietario, ...)
  - ...
- Espacio y estructuras de nivel interno
- Integridad de objetos
- Usuarios
- Privilegios y roles
- Información para Auditoría
- Información sobre la BD (tablespaces, datafiles, etc.)

- Al ser tablas e índices, se manejan y consultan como todas las demás (temas 1 y 2)
- Los objetos estan en el tablespace SYSTEM
- La información se almacena en:
  - Tablas: denominadas tablas base, sólo accesibles al SGBD y el usuario sys (¡no se recomienda!)
  - Vistas: sirven para el acceso simple, selectivo y personalizado a la información de las tablas base y están accesibles a cualquier usuario (según sus permisos)

- Los nombres son importantes:
  - **USER**_... determina que la vista contiene información sobre los objetos del usuario que ejecuta la consulta
  - **ALL**_... determina que la vista contiene información sobre todos los objetos accesibles para el usuario que ejecuta la consulta (incluye la información de USER_...)
  - **DBA**_... determina que la vista consulta la tabla de catálogo tal y como está almacenada (solo el usuario sys y otros con permiso pueden hacerlo)

- Organizar
- Mantener actualizada
- Conocer (útil para los tres niveles)

```
CREATE TABLE CARD (
    CARDID VARCHAR2(20) CONSTRAINT CARD_CARDID_PK PRIMARY KEY,
    CARDNAME VARCHAR2(30)
        CONSTRAINT CARD_CARNAME_NOTNULL NOT NULL,
    ACCOUNTNO VARCHAR2(20)
        CONSTRAINT CARD_ACCOUNTNO_NOTNULL NOT NULL
        CONSTRAINT CARD_ACCOUNTNO_FK REFERENCES ACCOUNT,
    EXPDATE DATE CONSTRAINT CARD_EXPDATE_NOTNULL NOT NULL,
    DAILYLIMIT NUMBER(4)
        CONSTRAINT CARD_DAILYLIMIT_NOTNULL NOT NULL
        CONSTRAINT CARD_DAILYLIMIT_POSSITIVE
            CHECK DAILYLIMIT >= 0,
    LASTLIMIT NUMBER(6,2)
        CONSTRAINT CARD_LASTLIMIT_NOTNULL NOT NULL
        CONSTRAINT CARD_LASTLIMIT_POSSITIVE CHECK LASTLIMIT >= 0
        AND
        CONSTRAINT CARD_LAST_LIMIT_LESSTHANDAILY
            CHECK LASTLIMIT <= DAILYLIMIT
);
```

Objeto

```
CREATE TABLE CARD (
    CARDID VARCHAR2(20) CONSTRAINT CARD_CARDID_PK PRIMARY KEY,
    CARDNAME VARCHAR2(30)
        CONSTRAINT CARD_CARNAME_NOTNULL NOT NULL,
    ACCOUNTNO VARCHAR2(20)
        CONSTRAINT CARD_ACCOUNTNO_NOTNULL NOT NULL
        CONSTRAINT CARD_ACCOUNTNO_FK REFERENCES ACCOUNT,
    EXPDATE DATE CONSTRAINT CARD_EXPDATE_NOTNULL NOT NULL,
    DAILYLIMIT NUMBER(4)
        CONSTRAINT CARD_DAILYLIMIT_NOTNULL NOT NULL
        CONSTRAINT CARD_DAILYLIMIT_POSSITIVE
            CHECK DAILYLIMIT >= 0,
    LASTLIMIT NUMBER(6,2)
        CONSTRAINT CARD_LASTLIMIT_NOTNULL NOT NULL
        CONSTRAINT CARD_LASTLIMIT_POSSITIVE CHECK LASTLIMIT >= 0
        AND
        CONSTRAINT CARD_LAST_LIMIT_LESSTHANDAILY
            CHECK LASTLIMIT <= DAILYLIMIT
);
```

```
CREATE TABLE CARD (
    CARDID VARCHAR2(20) CONSTRAINT CARD_CARDID_PK PRIMARY KEY,
    CARDNAME VARCHAR2(30)
        CONSTRAINT CARD_CARNAME_NOTNULL NOT NULL,
    ACCOUNTNO VARCHAR2(20)
        CONSTRAINT CARD_ACCOUNTNO_NOTNULL NOT NULL
        CONSTRAINT CARD_ACCOUNTNO_FK REFERENCES ACCOUNT,
    EXPDATE DATE CONSTRAINT CARD_EXPDATE_NOTNULL NOT NULL,
    DAILYLIMIT NUMBER(4)
        CONSTRAINT CARD_DAILYLIMIT_NOTNULL NOT NULL
        CONSTRAINT CARD_DAILYLIMIT_POSSITIVE
            CHECK DAILYLIMIT >= 0,
    LASTLIMIT NUMBER(6,2)
        CONSTRAINT CARD_LASTLIMIT_NOTNULL NOT NULL
        CONSTRAINT CARD_LASTLIMIT_POSSITIVE CHECK LASTLIMIT >= 0
        AND
        CONSTRAINT CARD_LAST_LIMIT_LESSTHANDAILY
            CHECK LASTLIMIT <= DAILYLIMIT
);
```

Tabla

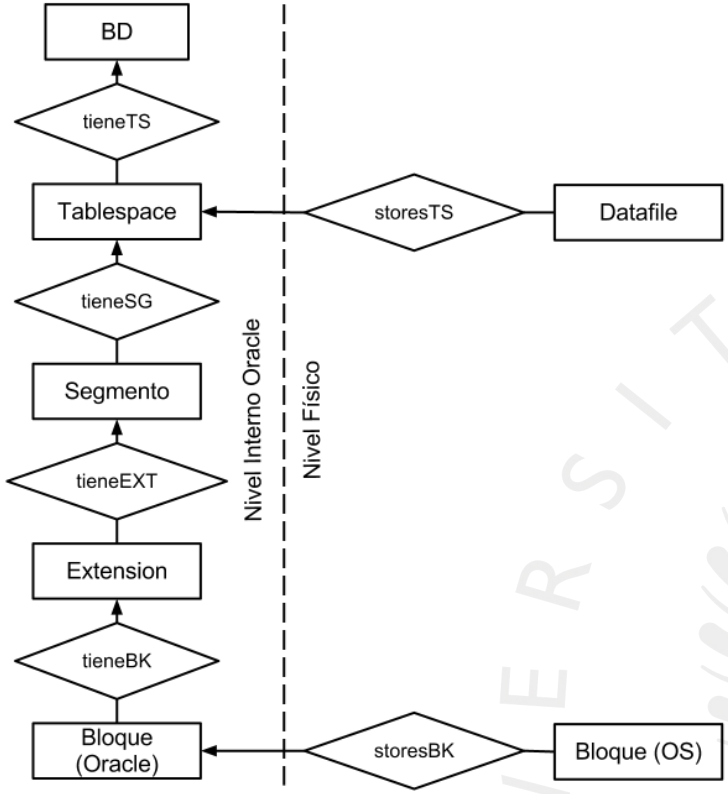
```
CREATE TABLE CARD (
    CARDID VARCHAR2(20) CONSTRAINT CARD_CARDID_PK PRIMARY KEY,
    CARDNAME VARCHAR2(30)
        CONSTRAINT CARD_CARNAME_NOTNULL NOT NULL,
    ACCOUNTNO VARCHAR2(20)
        CONSTRAINT CARD_ACCOUNTNO_NOTNULL NOT NULL
        CONSTRAINT CARD_ACCOUNTNO_FK REFERENCES ACCOUNT,
    EXPDATE DATE CONSTRAINT CARD_EXPDATE_NOTNULL NOT NULL,
    DAILYLIMIT NUMBER(4)
        CONSTRAINT CARD_DAILYLIMIT_NOTNULL NOT NULL
        CONSTRAINT CARD_DAILYLIMIT_POSSITIVE
            CHECK DAILYLIMIT >= 0,
    LASTLIMIT NUMBER(6,2)
        CONSTRAINT CARD_LASTLIMIT_NOTNULL NOT NULL
        CONSTRAINT CARD_LASTLIMIT_POSSITIVE CHECK LASTLIMIT >= 0
        AND
        CONSTRAINT CARD_LAST_LIMIT_LESSTHANDAILY
            CHECK LASTLIMIT <= DAILYLIMIT
);
```

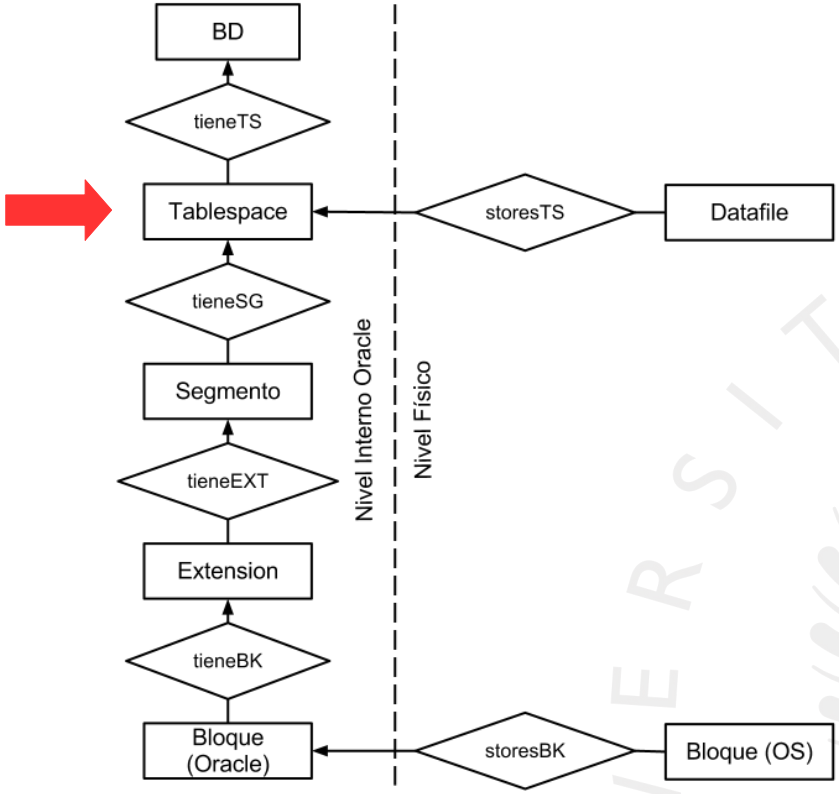
Columnas

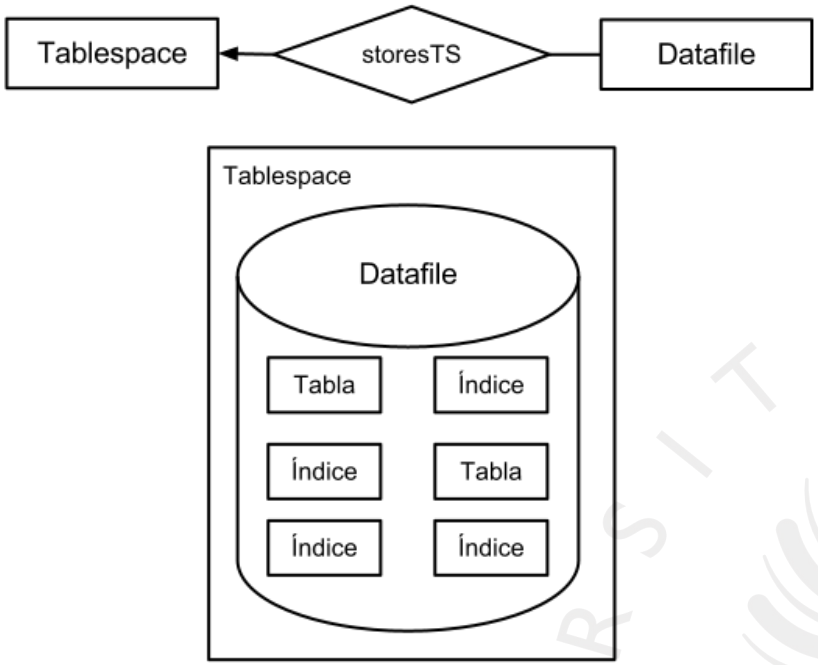
```
CREATE TABLE CARD (
    CARDID VARCHAR2(20) CONSTRAINT CARD_CARDID_PK PRIMARY KEY,
    CARDNAME VARCHAR2(30)
        CONSTRAINT CARD_CARNAME_NOTNULL NOT NULL,
    ACCOUNTNO VARCHAR2(20)
        CONSTRAINT CARD_ACCOUNTNO_NOTNULL NOT NULL
        CONSTRAINT CARD_ACCOUNTNO_FK REFERENCES ACCOUNT,
    EXPDATE DATE CONSTRAINT CARD_EXPDATE_NOTNULL NOT NULL,
    DAILYLIMIT NUMBER(4)
        CONSTRAINT CARD_DAILYLIMIT_NOTNULL NOT NULL
        CONSTRAINT CARD_DAILYLIMIT_POSSITIVE
            CHECK DAILYLIMIT >= 0,
    LASTLIMIT NUMBER(6,2)
        CONSTRAINT CARD_LASTLIMIT_NOTNULL NOT NULL
        CONSTRAINT CARD_LASTLIMIT_POSSITIVE CHECK LASTLIMIT >= 0
        AND
        CONSTRAINT CARD_LAST_LIMIT_LESSTHANDAILY
            CHECK LASTLIMIT <= DAILYLIMIT
);
```

Restricciones

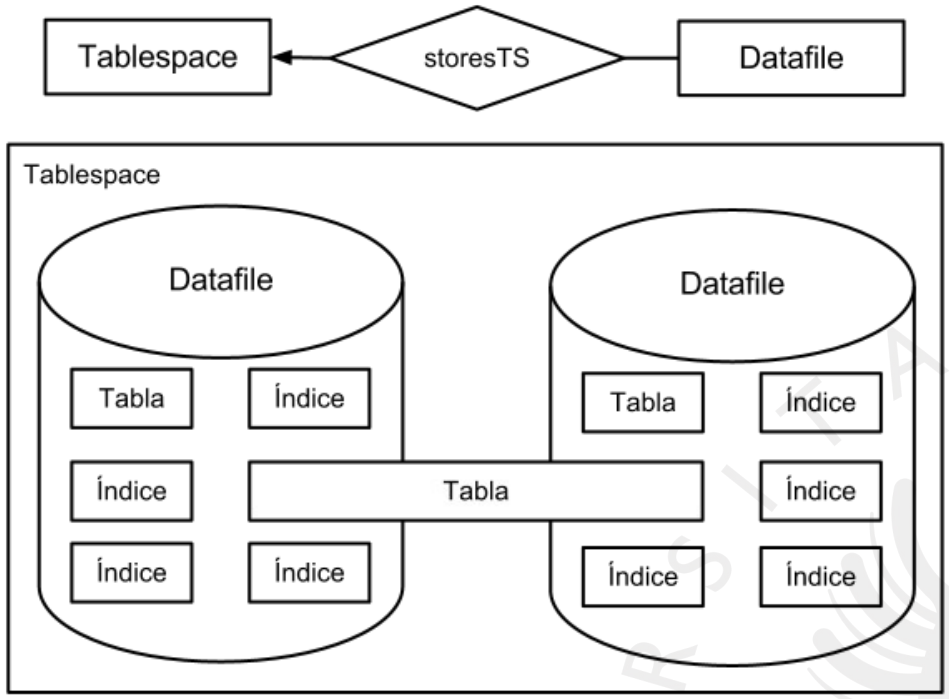
- Es habitual un fichero “con todo”
- Los bloques se agrupan para recuperar información relacionada fácilmente.
- A nivel físico de disco y S. O. sólo hay bloques y ficheros.
- Al nivel físico de Oracle® hay: tablespaces, segmentos, extensiones y bloques


BD
tieneTS
Tablespace
storesTS
Datafile
tieneSG
Segmento
tieneEXT
Extension
tieneBK
Bloque (Oracle)
storesBK
Bloque (OS)
Nivel Interno Oracle
Nivel Físico

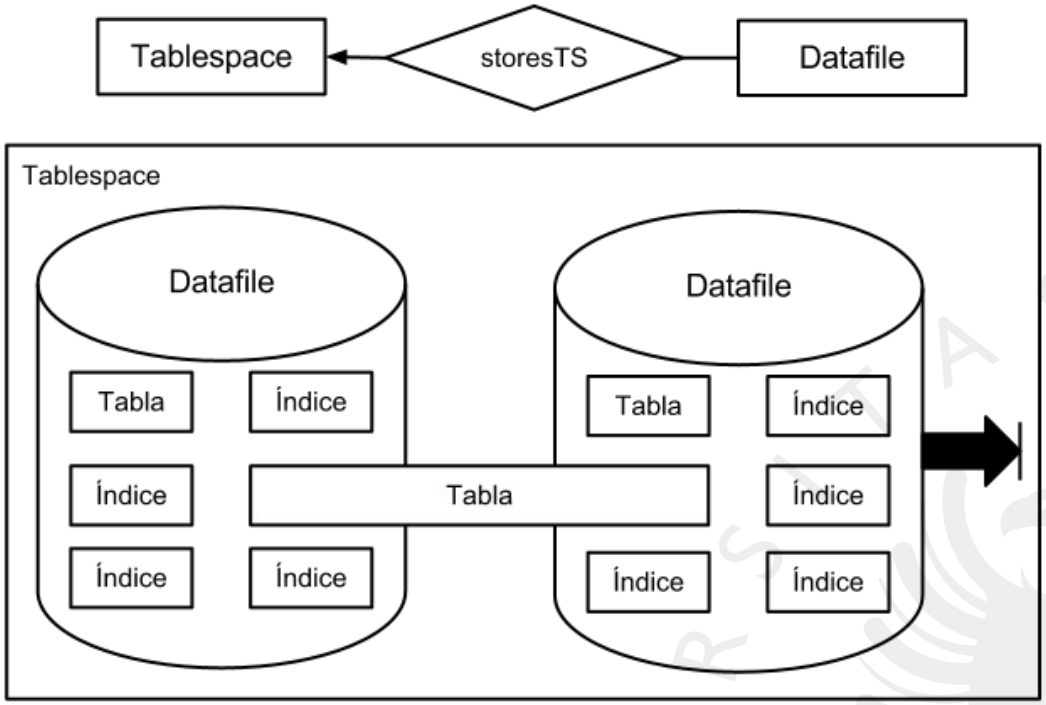




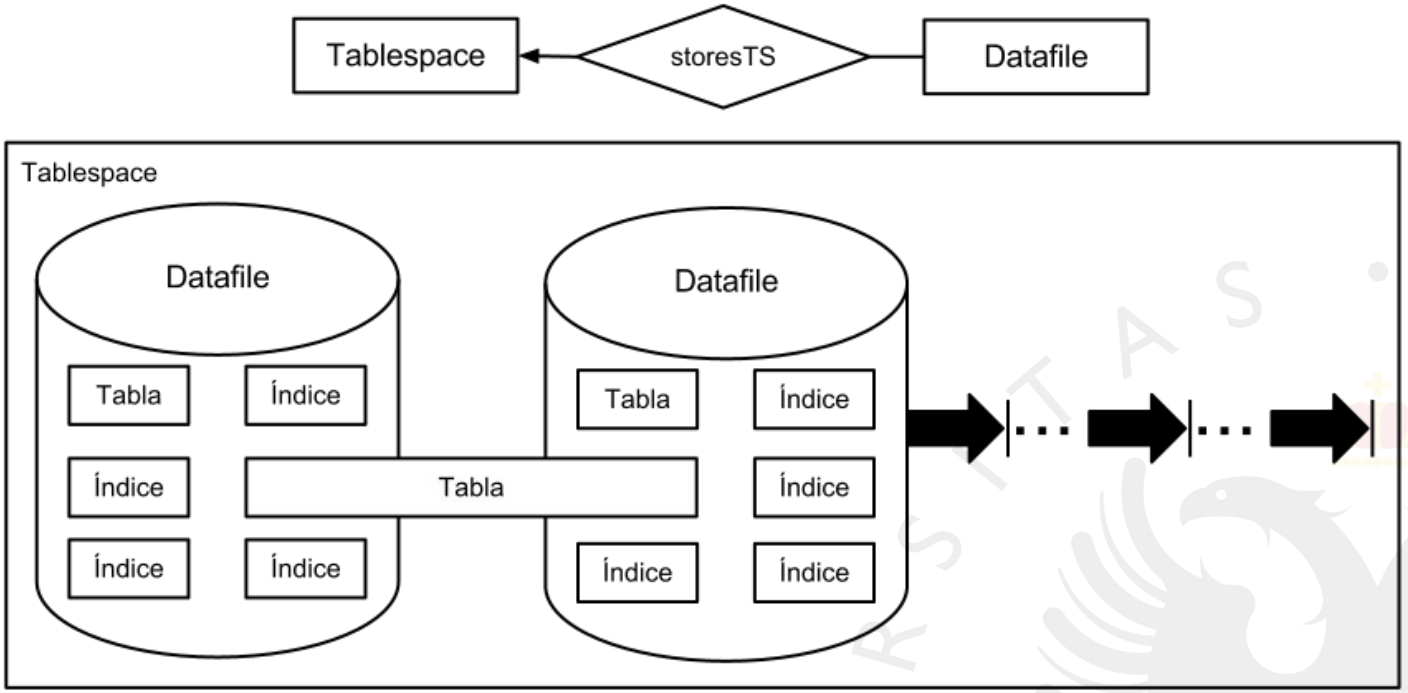

```
CREATE TABLESPACE users
   DATAFILE 'c:\oracle\oradata\users01.dbf'
   SIZE 20M;
```



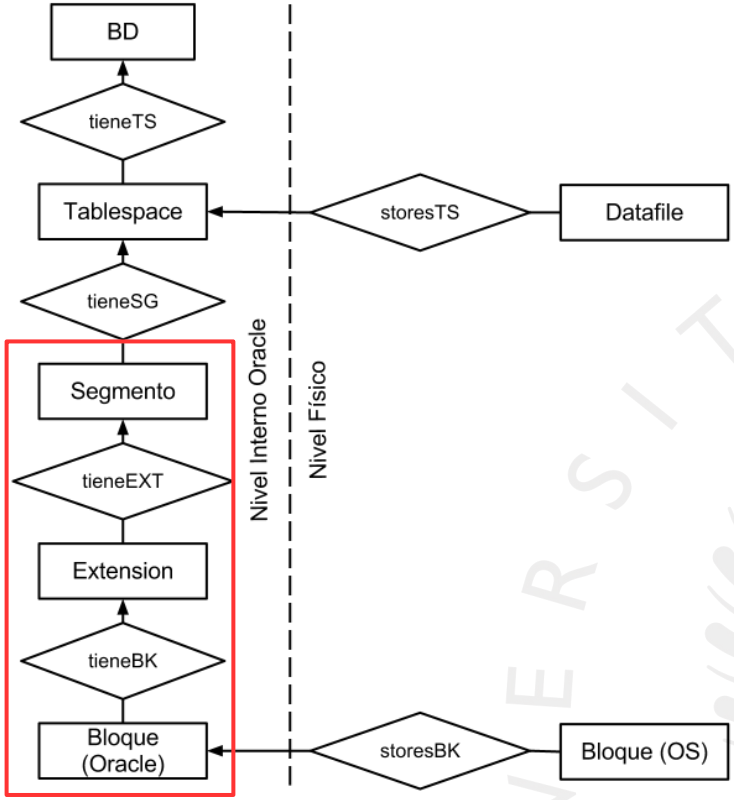

```
ALTER TABLESPACE users
   ADD DATAFILE 'c:\oracle\oradata\users02.dbf'
   SIZE 20M;
```



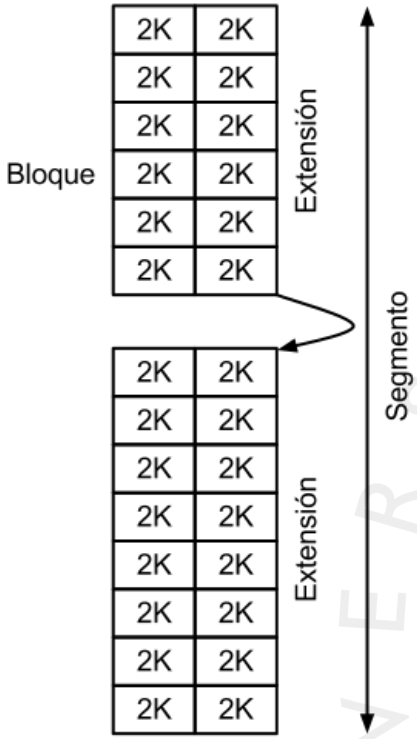

```
ALTER DATABASE DATAFILE
   'c:\oracle\oradata\users02.dbf' RESIZE 15M;
```




```
ALTER DATABASE DATAFILE
   'c:\oracle\oradata\users02.dbf' AUTOEXTEND ON
   NEXT 15M MAXSIZE 100M;
```


BD
tieneTS
Tablespace
storesTS
Datafile
tieneSG
Segmento
tieneEXT
Extension
tieneBK
Bloque (Oracle)
storesBK
Bloque (OS)
Nivel Interno Oracle
Nivel Físico

- Cada vez que se crea una tabla, se crea un **segmento** que contiene una **extensión**.
- Las extensiones dentro de un segmento se enganchan con apuntadores y pueden ser de distintos tamaños.
- Cada extensión contiene un conjunto de bloques. Cuando la extensión se completa, se genera otra extensión encadenada en el segmento.


Bloque
2K
Extensión
Segmento
Extensión

- Tipos de segmentos:
  - de datos (tablas)
  - de índice (índices)
  - temporales (resultados intermedios de order by, group by, union, intersect, minus, ...)
  - rollback (valores antiguos de datos en update)

- Estructura del bloque:
  - Cabecera: dirección y tipo de segmento
  - Directorio de tablas que tienen tuplas en el bloque
  - Directorio de tuplas en este bloque incluida dirección
  - Zona de datos donde se almacenan los registros
  - Espacio libre

- El conjunto de objetos de un usuario se denomina **esquema** y está compuesto por estructuras lógicas como:
  - tablas, vistas, índices, clústers,
  - procedimientos, funciones, paquetes,
  - disparadores,
  - ...

- Estructura lógica con columnas (atributos, identificados por nombre) y filas (tuplas, identificadas por contenido)
- Al crear una tabla, se crea un segmento con una extensión, y se adjudica a un tablespace por defecto salvo que se especifique
- Si una fila no cabe en un bloque se genera otra bloque en la extensión, enlazado al anterior, y la tupla se parte.

- Cada fila tiene una cabecera con información de cada fragmento, apuntadores, columnas en cada parte, tamaños, etc.
- Una fila tiene un rowid único e invariable.
- Algunos sistemas admiten el tipo tabla como tipo para una columna.

```
CREATE TABLE CARD (
   ...
) TABLESPACE users;
```

- Para asignar la tabla creada a un tablespace concreto.

```
CREATE TABLE CARD (
   ...
) TABLESPACE users
STORAGE (INITIAL 100K NEXT 100K MAXEXTENTS
10);
```

- Para asignar la tabla creada a un tablespace concreto y
- crear un segmento con una extensión inicial de 100KB para la primera extensión, las siguientes de otros 100KB hasta llegar a un número máximo de 10 extensiones.

```
DROP TABLE CARD;
```

- Presentación de datos procedentes de una o más tablas o vistas.
- Es como guardar una consulta.
- Se crean mediante el comando CREATE VIEW.
- Para consultar las vistas, se consulta la vista DBA_VIEWS.

- Generalmente, no son actualizables, salvo que cumplan ciertas restricciones:
  - No pueden incluir agrupadores o agregaciones
  - No puede incluir la cláusula DISTINCT
  - No puede incluir la reunión ni operadores de conjuntos
  - Todos los atributos con restricción NOT NULL (incluida la clave) deben estar en la vista
- Se elimina mediante la sentencia DROP VIEW

- Usos:
  - Seguridad (ocultar tuplas o atributos)
  - Abstraer de la complejidad de la estructuración de los datos
  - Simplificar comandos
  - Aislan aplicaciones de los cambios (si lo que contiene la vista no cambia, claro)
  - Para consultas complejas
  - Para consultas complejas que se usan mucho.

- Las vistas son el principal mecanismo para implementar el nivel externo propuesto por la arquitectura ANSI/SPARC

- Agilizan el acceso pero ocupan espacio
- Enlentecen las inserciones y modificaciones
- Hay que pensarlo dos veces antes de crearlos
- Oracle® crea un índice para la clave (¡no crees otro!)

- Se usan para:
  - buscar registros por valores específicos (en columnas indexadas)
  - recorrer una tabla en orden distinto al físico (ORDER BY)
  - buscar registros en un rango (en columna indexada)
- La sentencia para crearlos es CREATE INDEX
- Después de crear la tabla, mejor insertar las tuplas y después crear el índice.
- La indexación típica suele ser un árbol B* equilibrado, aunque puede usar bitmaps o hash.

- Índices con más de un atributo son útiles cuando se consulta por los valores de esos atributos de manera ordenada o, en caso de consultarse menos, se consultan desde el primero en adelante.
- Es necesario mantenerlos. Bórralos si:
  - Ya no sirven
  - No mejora la eficiencia
  - Hay que cambiar los campos que se indexan
  - Hay que rehacerlo
- Para borrarlo, se usa la sentencia DROP INDEX

- Vistas de catálogo: DBA_INDEXES y DBA_IND_COLUMNS

- Para el almacenamiento cercano de tablas que comparten campos y a las que se accede de forma conjunta (reunión natural).
- Si se accede a cada tabla individualmente y de forma frecuente, no es eficiente.
- Si se modifican frecuentemente las tablas (en tuplas) no es eficiente.
- Mejoran la reunión natural al reducir los accesos a disco.
- El campo o campos de reunión se almacenan una sola vez.

- Creación:
  - Primero, se crea el clúster con la sentencia CREATE CLUSTER
  - Después, se crea cada tabla con su correspondiente CREATE TABLE:

```
CREATE TABLE <nombre> (
    ...
) CLUSTER <nombre> (<campos de reunión>);
```

  - Antes de insertar datos, crear el índice sobre los campos del clúster:

```
CREATE INDEX <nombre> ON CLUSTER <nombre>;
```

- Destrucción con la sentencia DROP CLUSTER
- Para borrar:
  - Hay que borrar las tablas primero o usar la cláusula INCLUDING TABLES
  - Hay que borrar llaves externas que las referencian salvo que se incluya la cláusula CASCADE CONSTRAINTS
  - Se puede borrar su índice con DROP INDEX pero no se podrá acceder al contenido del clúster si no se reconstruye dicho índice