FICHEROS y BB.DD.

Práctica 3

Diseño Físico en Oracle®

Análisis del Problema

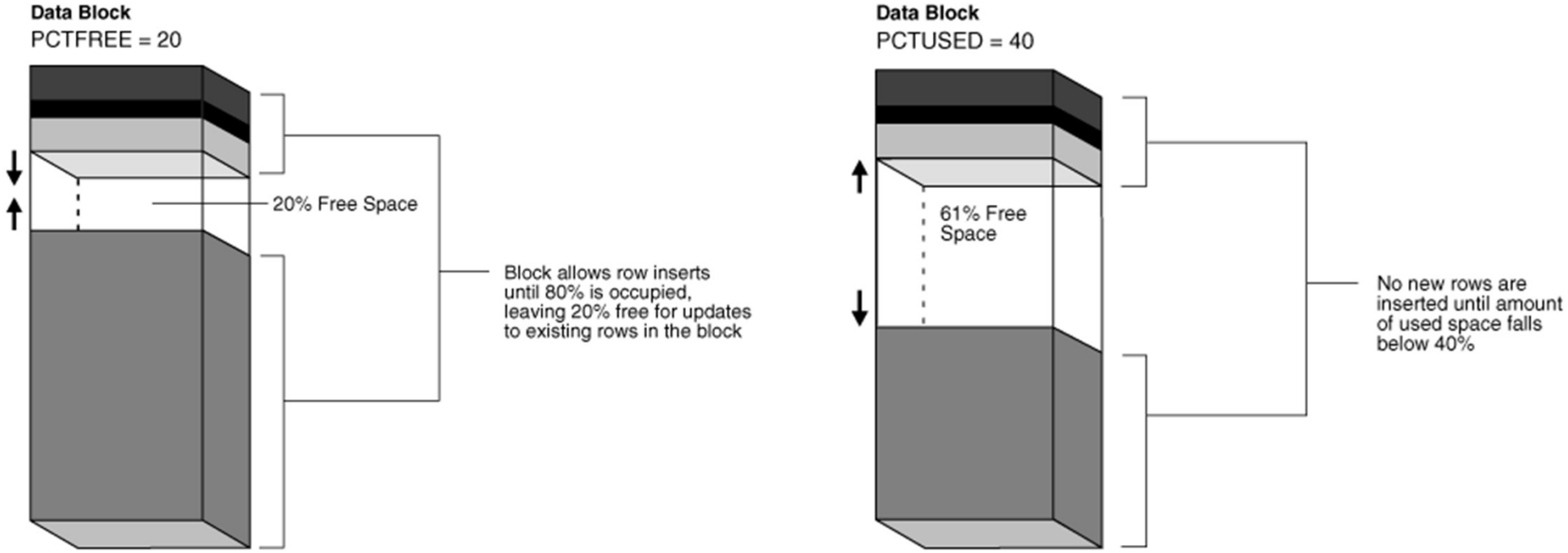
- Observar **situación actual**: estructuras (volumen), organización, procesos (coste individual y frecuencias), claves de proyección, claves de selección (cardinalidad dominios), combinaciones,...
- Proponer **mejoras** para cada proceso
- Analizar el **impacto** de cada mejora sobre el resto de procesos
- Realizar una **propuesta** final → diseño físico
  - Parametrización (parámetros físicos de cubo, ~~archivo, sistema~~...)
  - Organizaciones base (clusters)
  - Estructuras auxiliares
  - Otras mejoras (hints, reescritura consultas y procedimientos, ...)
- **Evaluar**

Diseño Físico

- Algunos parámetros ya están establecidos, por ejemplo:
  1. Espacio de cubo (DB block size = 8KB); se define para cada *tablespace*, y por ende para los objetos creados en él.
  2. Los ficheros de datos son seriales no-consecutivos
  3. El espacio libre distribuido es PCTFREE=10 & PCTUSED=60, pero puede cambiarse para cada segmento (objeto).
- Para adoptar las mejores decisiones, deben conocerse los parámetros, por ejemplo (de las *estadísticas de estado*):
  - `USER_TABLES.AVG_ROW_LEN`: tamaño (medio) del registro
  - `USER_TABLES.NUM_ROWS`: número de registros (*r*)
  - `USER_TABLES.BLOCKS`: número de cubos (N)

Diseño Físico: DB-blocks en Oracle

Ejemplo de cubo en ORACLE ® con PCFREE y PCUSED



- El espacio de cubo (DB-block size) está asociado al *tablespace*

Diseño Físico: Tablespaces

- En nuestra instancia de BD hemos creado varios *tablespaces* para poder trabajar con diferentes tamaños de cubo

```
CREATE TABLESPACE TAB_2k DATAFILE 'TAB_2k.dbf' BLOCKSIZE 2048;
CREATE TABLESPACE TAB_8k DATAFILE 'TAB_8k.dbf' BLOCKSIZE 8192; -- default
CREATE TABLESPACE TAB_16k DATAFILE 'TAB_16k.dbf' BLOCKSIZE 16384;
```

- Se puede indicar el *tablespace* donde se quiere almacenar una tabla, índice, cluster..

```
CREATE … TABLESPACE tu_tablespace;
```

- Algunos objetos Oracle se pueden mover a otro *tablespace*.

```
ALTER INDEX nombre_ind REBUILD TABLESPACE nuevo_tablespace;
```

Propiedades Lógicas y Físicas

- Al crear un *objeto físico* (table, index, cluster, materialized view) se deben indicar sus propiedades lógicas (definición) y también se pueden añadir propiedades físicas:

```
CREATE objeto nombre (def.lógica…) prop.físicas;
```

```
Ejemplo: CREATE TABLE mytable (Atributo1  VARCHAR2(10), …)
                  PCTFREE 10
                  PCTUSED 60
                  TABLESPACE mi_tablespace
                  STORAGE(INITIAL 5);
```

- Si no se especifican, se adoptarán valores por defecto.

Selección de índices

- La selección de índices forma parte del diseño físico
- Consiste en decidir las estructuras auxiliares para optimizar el rendimiento de la BD de acuerdo a los procesos que la actualizan o consultan
- La sintaxis básica de creación de índices es:

```
CREATE [ind_type] INDEX ind_name ON
  table_name(ind_key) TABLESPACE name;
```

donde

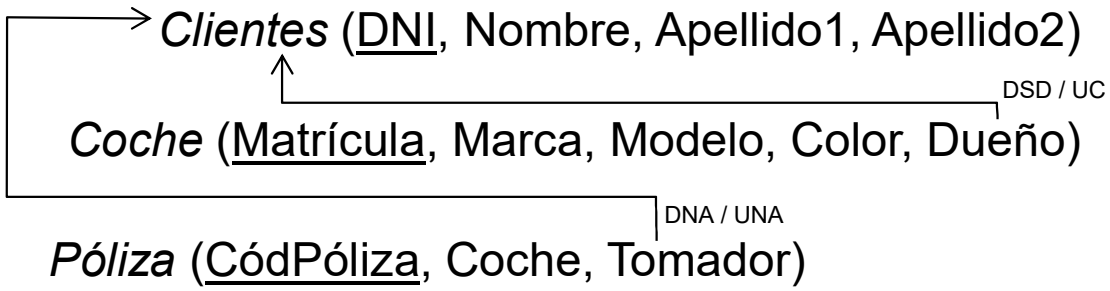

```
ind_type := UNIQUE | BITMAP |    (default)
ind_key := <columnas de tabla> (sep. por comas)
```

Clusters en Oracle®

- En ORACLE®, un *cluster* es la definición de clave privilegiada.
- Las tablas de un *cluster* se almacenan conjuntamente (**toda la fila** combinada se almacena físicamente **en el mismo cubo**). Beneficia JOIN y acceso por la clave privilegiada, perjudica a todo lo demás.
- La elección de la **clave** es **crítica**: puede bajar la densidad.
- Se puede redefinir el tamaño de cubo (celdas).
- *Se puede hacer un cluster mono-tabla* (cambia organización base)
- El *cluster* puede ser **indizado** o **disperso** (con **ordenación** opcional) para mejorar la **densidad**, los **procesos selectivos**, y los **procesos ordenados**, respectivamente).
- El cluster debe **crearse antes de crear la tabla**
  http://docs.oracle.com/cd/B28359_01/server.111/b28286/statements_5001.htm#i2105031

Implementación de Clusters

Ejemplo:




```
CREATE CLUSTER identidad (DNI VARCHAR2(9)) TABLESPACE users;
```

Por defecto, el cluster es indizado

Añadimos las tablas la cluster

```
CREATE TABLE cliente(…) CLUSTER identidad (DNI);
CREATE TABLE coche(…) CLUSTER identidad (dueño);
CREATE TABLE poliza(…) CLUSTER identidad (tomador);
CREATE INDEX ind_dni ON CLUSTER identidad TABLESPACE users;
```

```
identidad
( DNI C(9),
  cliente (nombre C(25), apellido1 C(15), apellido2 C(15)),
  coche (matrícula C(7), marca C(20), modelo C(20),color C(10) )*,
  poliza (cod C(30), coche C(7) )*
);
```

Implementación de Clusters

Sintaxis de creación de cluster:

```
CREATE  CLUSTER clustername (atributo tipo(tam)  [, atributo …])
        [PCTFREE XX]
        [PCTUSED YY]
        [SIZE]
        [TABLESPACE name]
        [STORAGE(…)]
        [INDEX
        | [SINGLE TABLE] HASHKEYS N [HASH IS …] ]  ;
```

Ejemplos de cluster:

```
CREATE  CLUSTER dni_clust(dni NUMBER(8)) INDEX;

CREATE  CLUSTER catalog_clust(num_ref NUMBER(10))
        STORAGE(BUFFER_POOL KEEP);

CREATE  CLUSTER student_clust(nia NUMBER(9))
        SINGLE TABLE HASHKEYS 797 HASH IS MOD(nia,10000);

CREATE  CLUSTER ident(DNI NUMBER(8), tlf NUMBER(9))
        HASHKEYS 503 HASH IS DNI+tlf;

CREATE  CLUSTER ident(DNI NUMBER(8), apellido VARCHAR2(15) SORT)
        HASHKEYS 100 HASH IS DNI;
```

Plan de Ejecución Previsto:

```
EXPLAIN PLAN SET statement_id = 'plan_name' FOR
   <dml_sentence>;


SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

SELECT PLAN_TABLE_OUTPUT
  FROM TABLE(DBMS_XPLAN.DISPLAY(NULL,'plan_name','BASIC'));


SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY(FORMAT=>'+ALLSTATS'));
```

- con privilegio select en ...
  `V_$SESSION , V_$SQL , V_$SQL_PLAN , V_$SQL_PLAN_STATISTICS_ALL`

Plan de Ejecución Efectivo: Autotrace

- La traza proporciona estadísticas (costes) y plan:

1. **`autotrace on`**: plan + resultado+ estadísticas
2. **`autotrace on statistics`**: resultado+ estadísticas
3. **`autotrace on explain`**: plan + resultado
4. **`autotrace traceonly`**: plan + estadísticas
5. **`autotrace off`**: desactivar

Ejemplo:

```
set serveroutput on
set timing on
set autotrace on
```

http://docs.oracle.com/cd/B28359_01/server.111/b28274/ex_plan.htm#i18300

(tabla 12 todas las operaciones)

Medidas de rendimiento

- Las vistas de estadísticas proporcionan información:
  1. Accediendo a Ora Stats (a través del paquete dado)

```
set serveroutput on
begin
PKG_COSTES.RUN_TEST(10); --Número de repeticiones
end;
```

NOTA: las estadísticas de Oracle® pueden dar distintos valores para la misma carga. Por lo que se recomienda ejecutar varias veces la carga (coger la media)

Optimización basada en... *Reglas* vs *Costes*

- El plan de ejecución en Oracle se puede obtener por dos métodos, y se puede dirigir por Hints:
  - REGLAS: decisiones tomadas en base a criterios apriorísticos.
  - **COSTES**: las decisiones se toman calculando la mejor opción. Camino eficiente si las *estadísticas de estado de la BD* son fiables.
    - la fiabilidad dependerá de la significatividad del muestreo, de la frecuencia de actualización, etc.
    - Para actualizar estadísticas (for table & for indexes):

```
ANALYZE TABLE nombre COMPUTE STATISTICS;

ANALYZE TABLE nombre ESTIMATE STATISTICS SAMPLE 10 PERCENT;
```

- **DIRIGIDO** por el usuario: el usuario conoce la semántica, uso, ... Eventualmente, puede decidir mejor. Para ello, Oracle tiene los *Hints*.

HINTS en Oracle®

- Ningún optimizador es infalible: las reglas son generales; los costes dependen de la actualización de parámetros; las estadísticas son rígidas; y todos esos métodos ignoran la semántica de los datos.
- **HINTS**: podas del árbol de decisión, marcados por el usuario.
- No estándar, en PL/SQL se especifican como comentarios:

```
SELECT /*+ HINT */ attributes FROM tablename … ;

SELECT --+ HINT
   attributes FROM tablename … ;
```

- Se pueden especificar varios HINTS para la misma instrucción (separados por espacios).
- Algunos HINTS tienen versión opuesta (ejemplo: INDEX→ NOINDEX)

Tema 8.3.5: HINTs Oracle® más usuales

Sintaxis del HINT			Descripción
`/*+RULE */`	`/*+ALL_ROWS*/`	`/*+FIRST_ROWS(n)*/`	elige tipo de optimizador (reglas/costes)
`/*+ FULL(tablename) */`			recorrido a totalidad (*full scan*) de tabla
`/*+ ROWID(tablename) */`			rowid scan (cubos individuales)
`/*+ CLUSTER(tablename) */`			tabla en cluster accedida por el mismo
`/*+ HASH(tablename) */`			usa la dispersión de una tabla en *cluster*
`/*+ ORDERED */`	`/*+ LEADING(tab1 tab2)*/`		join tables in Q order, or specify an order
`/*+USE_NL(t)*/`	`/*+USE_MERGE(t)*/`	`/*+USE_HASH(t)*/`	join tables with nested loops/merge/hash (*inner*)
`/*+ INDEX(tablename) */`	`/*+NO_INDEX(table)*/`		use/forbids any index on the given table
`/*+ INDEX(tablename index1 index2 ... ) */`			use/forbids specific index/es (one or +)
`/*+ AND_EQUAL(tablename index1 index2 […]) */`			use more than one index (up to 5)
`/*+ INDEX_ASC(…) */`	`/*+ INDEX_DESC(…) */`		index range scan in asc/desc order
`/*+ INDEX_FFS(…) */`	`/*+ INDEX_SS(…) */`		index fast full scan // index skip scan
`/*+ INDEX_JOIN(tablename [indexname […]]) */`			join indexes (sort of inverted access)
`/* CACHE */`	`/* NOCACHE */`		situar cubos al comienzo/final de la lista LRU
`insert /*+ APPEND */ …`	`insert /*+NOAPPEND*/`		escritura directa HWM (~~buffer, stack, RI, trigger~~)

Guía para la práctica 3: Diseño físico

- Pasos para realizar la práctica (entrega semana 14):
  1. Preparar la BD
     a. Ejecuta el script de creación
     b. Ejecuta el script de carga
  2. Medir la eficiencia (ejecuta `PKG_COSTES.RUN_TEST`)
  3. Plantear e implementar un diseño físico
  4. Medir su eficiencia
  5. Escribir la memoria
     a. Introducción y análisis del problema
     b. Diseño físico
     c. Resultados de la evaluación (comparativa)
     d. Conclusiones

Ejercicios (1/2) – Plan de Ej. + Opt. SQL

1. Obtén el plan de ejecución y los 'consistent gets' de esta consulta:

```
select distinct DNI, performer, when
   from clients C, attendances A
   where UPPER(C.e_mail)='ESPANIARD@CLIENTS.VINYLINC.COM'
         AND UPPER(A.client)=UPPER(C.e_mail);
```

2. Compáralos con esta otra versión:

```
select distinct DNI, performer, when
   from clients C JOIN attendances A ON (A.client=C.e_mail)
   where UPPER(C.e_mail)='ESPANIARD@CLIENTS.VINYLINC.COM';
```

3. Y con esta otra:

```
select distinct DNI, performer, when
  from (select DNI, e_mail client from clients
          where UPPER(e_mail)='ESPANIARD@CLIENTS.VINYLINC.COM')
        join (select performer, when, client from attendances
          where UPPER(client)='ESPANIARD@CLIENTS.VINYLINC.COM')
using(client);
```

Ejercicios (2/2) – Indexes & …+

4. Crea un índice para el atributo *spic* sobre la tabla controls, y ejecuta la última versión de la consulta:

```
create index ind2 on attendances(client);
-- no effect!!!
```

5. Adapta la consulta y repite la ejecución. ¿en qué cambia?

```
select distinct DNI, performer, when
  from (select DNI, e_mail client from clients
          where e_mail='espaniard@clients.vinylinc.com')
        join (select performer,when,client from attendances
          where client='espaniard@clients.vinylinc.com')
using(client);
```

6. ¿Y si en lugar de cambiar la consulta, cambiamos el índice?:

```
drop index ind2;
create index ind2 on attendances(UPPER(client));
```