

paquetes. Todas ellas son estructuras lógicas de almacenamiento (a nivel conceptual) y no tienen una correspondencia física directa, aunque sí afectan en gran medida a las estructuras empleadas en los niveles interno y físico. De hecho, un tablespace puede contener objetos de diferentes esquemas y los objetos de un esquema pueden alojarse en diferentes tablespaces.

◇ *Las estructuras lógicas que maneja el SGBD Oracle son:*

- Tablas.
- Vistas.
- Índices.
- Clústers.

3.4.1. TABLAS

La **tabla** es la unidad básica de almacenamiento a nivel lógico y se compone de filas y columnas. Tablas y columnas se identifican por su nombre, pero las filas lo hacen por su contenido⁶.

◇ *Algunas de las características más importantes de las tablas son las siguientes:*

- Cuando se crea una tabla, automáticamente se crea un segmento (compuesto en principio por una sola extensión) para los futuros datos que se vayan incluyendo en ella. ↗ con un solo bloque
- Cada tabla pertenece a un tablespace concreto que, o bien se especifica en la sentencia de su creación o, en caso contrario, se adjudica uno por defecto.
- Las filas o tuplas que sobrepasan el tamaño de un bloque se parten y cada trozo se almacena en un bloque que es encadenado al anterior mediante el uso de un apuntador⁷.

⁶En la mayoría de los SGBD también existe una numeración interna para las filas (rowid o RID), aunque ésta no guarda ninguna relación con los datos que contiene.

⁷Emplea bloqueo partido con registros de longitud variable.

- Cada fila tiene una cabecera que guarda información de cada fragmento, punteros de encadenamiento, columnas de cada parte, tamaños, etc.
- Cada fila está identificada por un número o *rowid* que no varía desde que es introducida en la BD hasta que es eliminada.
- En algunos sistemas se puede crear una tabla con una columna de tipo tabla. Esta estructura recibe el nombre de *nested table* o tabla anidada.

En SQL, la creación de una tabla se hace con el comando CREATE TABLE según se muestra en el siguiente ejemplo:

```
CREATE TABLE cliente (
    clienteid      NUMBER(6) PRIMARY KEY,
    nombre          VARCHAR2(45),
    direc           VARCHAR2(40),
    ciudad          VARCHAR2(30)
) TABLESPACE users;
```

donde, como se ve, se debe especificar su nombre, los nombres, tipos y restricciones (si las hay) de los atributos y el nombre del tablespace en el que se ubicará la tabla. Esta última opción no es necesaria si el usuario que crea la tabla tiene asociado un tablespace por defecto para todos sus objetos.

Si queremos, además, especificar cómo debe realizarse el almacenamiento interno para una determinada tabla, la sentencia podría extenderse como sigue:

```
CREATE TABLE cliente (
    clienteid      NUMBER(6) PRIMARY KEY,
    nombre          VARCHAR2(45),
    direc           VARCHAR2(40),
    ciudad          VARCHAR2(30)
) TABLESPACE users
STORAGE (INITIAL 100K NEXT 100K MAXEXTENTS 10);
```

donde INITIAL indica el tamaño de la primera extensión del segmento y NEXT el tamaño de las siguientes, hasta reservar un máximo de MAXEXTENTS extensiones.

Por último, la eliminación de una tabla se hará mediante la sentencia

```
DROP TABLE cliente;
```

que liberará el espacio ocupado por el segmento correspondiente y eliminará todas las entradas del catálogo que hagan referencia a ella.

3.4.2. VISTAS

Una vista es una presentación de datos provenientes de una o más tablas, hecha a la medida de un usuario. Básicamente, consiste en asignar un nombre a la salida de una consulta y utilizarla como si de una tabla almacenada se tratara. De hecho, pueden usarse en lugar de cualquier nombre de tabla en los comandos del DSL. La vista es la estructura de más alto nivel dentro del nivel lógico y, de hecho, es el mecanismo básico de implementación del nivel externo, aunque Oracle no los diferencia formalmente hablando.

Salvo que se especifique lo contrario, las vistas no existen físicamente; su definición se almacena en el diccionario y éstas se reconstruyen cada vez que sea necesario. Por ello, sobre ellas no pueden aplicarse restricciones de integridad ni disparadores.

Gracias a las vistas podemos establecer niveles de seguridad adicionales a los que ofrece el sistema, ya que se puede ocultar cierta información y hacer visible a los usuarios solo la parte de la BD que necesiten para realizar sus tareas. Además, simplifican el aspecto de la BD y el uso de algunos comandos.

En SQL, la creación de una vista se hace mediante el comando CREATE VIEW, según se muestra en el siguiente ejemplo:

```
CREATE VIEW cli_Gr AS  
  SELECT clienteid, nombre, direc  
    FROM cliente  
   WHERE ciudad= 'Granada' ;
```

lo que básicamente produce la inserción de una fila en el catálogo, que se verá reflejada en la vista dba_views. Algunos de los atributos más interesantes de esta vista son los siguientes:

```
        dba_views(owner,view_name,text)  
y donde owner es el propietario de la vista, view_name el nombre que se le ha
```

asignado y text la sentencia *select* que permite reconstruirla. Si consultamos el catálogo con la sentencia

```
SELECT view_name, text
FROM dba_views
WHERE view_name='CLI_GR';
```

obtenemos el siguiente resultado:

VIEW_NAME	TEXT
CLI_GR	<pre>SELECT clienteid, nombre, dirección FROM cliente WHERE ciudad='Granada'</pre>

A partir de este momento, cualquier usuario autorizado podrá hacer uso de la vista cli_Gr como si de cualquier tabla se tratara. Por ejemplo, podríamos ejecutar sin problemas las siguientes consultas:

- *Encontrar los clientes de Granada con apellido Pérez.*
- ```
SELECT * FROM cli_Gr WHERE nombre LIKE '%PEREZ%'
```
- *Encontrar los nombres y las direcciones de los clientes de Granada que hayan hecho algún pedido a fecha de hoy.*
- ```
SELECT nombre, dirección
FROM cli_Gr c, pedido p
WHERE c.clienteid=p.clienteid and fechaped=SYSDATE;
```

Por su carácter virtual, existen fuertes *restricciones a la hora de insertar o actualizar datos en una vista* debido, principalmente, a que cada fila de una vista no siempre se corresponde con una fila de una tabla concreta. Cuando esto sucede, puede resultar imposible aplicar una modificación sobre una fila o un campo de una vista al no poderse encontrar el origen ni la ubicación real de la información a modificar.

Algunas de las restricciones más relevantes son:

- La definición de la vista no podrá incluir cláusulas de agrupamiento de tuplas (GROUP BY) o funciones de agregación (MAX, COUNT, AVG...) porque resumen los datos originales.

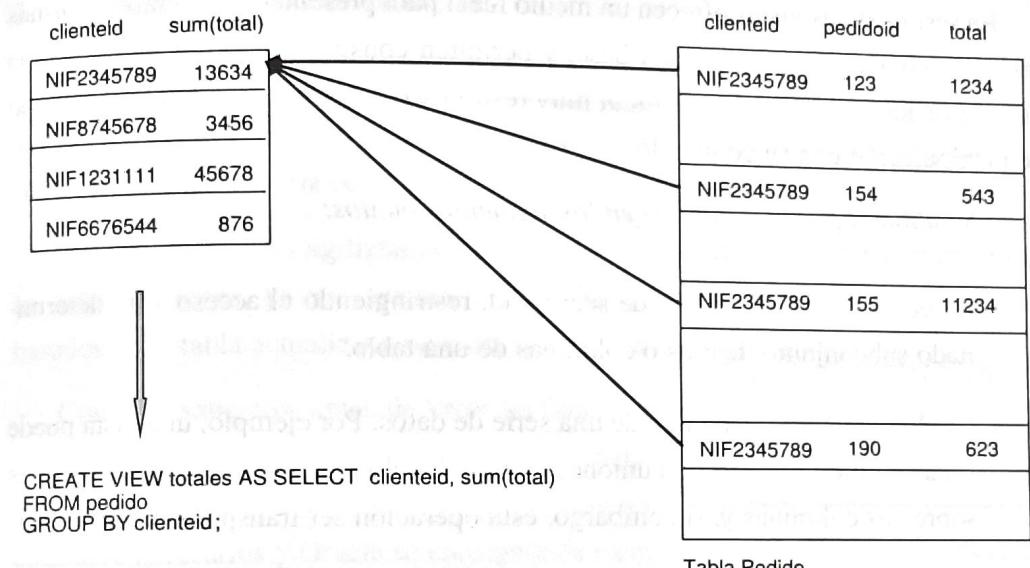


FIGURA 3.11 Vista no actualizable.

- La definición de la vista no podrá incluir la cláusula DISTINCT, para evitar que una misma fila en la vista se corresponda con más de una fila de la tabla base.
- La definición de la vista no podrá incluir operaciones de reunión ni de conjuntos, esto es, deberá construirse sobre una única tabla base.
- Todos los atributos que deban tomar siempre valor (NOT NULL y PRIMARY KEY) han de estar incluidos necesariamente en la definición de la vista. Piense el porqué.

Como puede verse, todas estas restricciones van encaminadas a evitar la ambigüedad que pudiera surgir si el sistema no encontrara una correspondencia única entre una tupla de la vista y una tupla de una tabla base, que es, en definitiva, donde se van a reflejar las modificaciones hechas sobre la vista.

Cuando una vista deja de ser útil, puede eliminarse mediante la sentencia

`DROP VIEW cli_gr;`

que solo afecta al contenido del catálogo.

En resumen, las vistas ofrecen un medio ideal para presentar de diferentes formas los datos contenidos en una tabla base, y permiten construir *subbases de datos* a la medida de los usuarios, pero resultan muy restrictivas a la hora de realizar algún tipo de manipulación con su contenido.

◊ *Su utilidad puede resumirse en los siguientes puntos:*

- Ofrecer un nivel adicional de seguridad, restringiendo el acceso a un determinado subconjunto de filas o columnas de una tabla.
- Ocultar la complejidad real de una serie de datos. Por ejemplo, una vista puede estar definida por varias reuniones (joins) de tablas con múltiples condiciones sobre sus columnas y, sin embargo, esta operación ser transparente al usuario.
- Simplificar los comandos al usuario.
- Presentar los datos desde diferentes perspectivas a como lo hace la tabla base. Aislara las aplicaciones de los cambios que se produzcan en las tablas base. Por ejemplo, si una vista referencia tres columnas de las cuatro que tiene la tabla base correspondiente y a esta última se le añade una columna más, ni la definición de la vista ni las aplicaciones que la utilicen, se verán afectadas.
- Para expresar consultas complejas que no pueden especificarse de otra manera. Éste es el caso, por ejemplo, de una vista que reúna una tabla con otra vista que se ha construido a partir de la unión de otras dos tablas.
- Para almacenar consultas complejas que son frecuentemente utilizadas. Por ejemplo, una consulta podría llevar a cabo gran cantidad de operaciones sobre los datos que contiene una determinada tabla. Si se salva esta consulta en forma de vista, las operaciones se llevarán a cabo cada vez que ésta sea utilizada.

Gracias a todas las características mencionadas, las vistas constituyen el principal mecanismo para implementar el nivel externo propuesto por la arquitectura ANSI/SPARC.

3.4.3. ÍNDICES

Como ya se explicó en profundidad en el Capítulo 1, los índices son creados para aumentar la eficiencia en los accesos a las tablas de la base de datos. A cambio, se paga con el espacio extra ocupado por éstos y el tiempo necesario para mantenerlos.

Aunque los índices agilizan la recuperación de los datos, sin embargo ralentizan las actualizaciones, ya que, cuando éstas se llevan a cabo, cada uno de los índices basados en la tabla actualizada necesita también una actualización.

Como ya sabemos, antes de crear un índice se debe determinar si es necesario, es decir, se debe estimar si el beneficio en el rendimiento que se obtendrá supera al costo derivado de su mantenimiento. El usuario solo debe crear aquellos índices que considere necesarios y Oracle se encargará de usarlos y mantenerlos automáticamente. El mantenimiento implica la modificación necesaria del índice cuando se añaden, modifican o eliminan registros de la tabla.

Cuando creamos una tabla, Oracle crea automáticamente un índice asociado a la clave primaria de la misma. No es conveniente crear índices sobre la clave primaria porque no solo no se consigue mejorar el rendimiento, sino que se empeora al tener que mantener un índice más. Por tanto, es importante crear los índices que se necesiten realmente y se deben borrar si no van a necesitarse de nuevo.

Oracle hace uso de los índices para incrementar el rendimiento cuando:

1. Busca registros por valores específicos de columnas indexadas.
2. Recorre una tabla en un orden distinto al orden en que se encuentran físicamente almacenados sus registros (order by).
3. Busca registros en un rango de valores de una columna indexada.

Para la creación de índices usaremos la sentencia CREATE INDEX cuya sintaxis básica es:

`CREATE INDEX nombre-del-índice ON tabla(atributo [ASC | DESC] ...);`

Por ejemplo, si queremos acelerar las consultas cuando busquemos a un cliente por su nombre, podemos crear un índice asociado al campo nombre de la tabla cliente.

`CREATE INDEX indice_cliente ON cliente(nombre);`

Cuando se crea una tabla es mejor insertar los registros antes de crear el índice. Si se crea antes el índice, Oracle deberá actualizarlo cada vez que se inserte un registro.

Cuando se crea un índice, Oracle recupera los campos indexados de la tabla y los ordena. A continuación almacena en una estructura especial (segmento de índice) los valores de los campos indexados junto con el identificador (ROWID) del registro correspondiente. Oracle usa por defecto árboles B* balanceados para igualar el tiempo necesario para acceder a cualquier fila, pero proporciona índices de otros tipos como mapa de bits (bitmap) y hash.

En la mayoría de los sistemas existe la posibilidad de crear índices compuestos, que son aquellos que se crean sobre más de un campo de la misma tabla. Los índices compuestos pueden acelerar la recuperación de información cuando la condición de la consulta correspondiente referencia a todos los campos indexados o solo al primero, por lo que el orden de las columnas usado en la definición del índice es importante; generalmente, los campos por los que se accede con mayor frecuencia se colocan antes en la creación del índice compuesto.

Por ejemplo, partiendo de la tabla emp con atributos empid, empnom, direc, localidad y deptoid, creamos un índice compuesto sobre los campos deptoid y empid:

`CREATE INDEX indice_emp ON emp(deptoid,empid);`

Dicho índice acelerará la recuperación de información de las consultas cuya condición incluya referencias al atributo deptoid o a los atributos deptoid y empid conjuntamente. Por ejemplo:

```
SELECT empnom FROM emp WHERE deptoid=15;
SELECT * FROM emp WHERE deptoid=10 and empid=127;
```

Sin embargo no mejorará el acceso cuando las consultas no incluyan referencias a los primeros campos del índice. Por ejemplo:

```
SELECT * FROM emp WHERE empid=32;
```

Como ya indicamos anteriormente, los índices no deben mantenerse cuando no son necesarios, y son varias las razones por las que puede interesar eliminarlos:

- Se creó con una finalidad concreta y ya no resulta de utilidad.
- Por las características de la tabla, el índice no mejora la eficiencia.
- Necesitamos cambiar los campos que se indexan.
- Necesitamos rehacer un índice muy fragmentado.

Para eliminar un índice usaremos la sentencia:

```
DROP INDEX nombre_del_index;
```

Se debe tener en cuenta que, cuando se borra una tabla, también se borran todos los índices asociados. Los índices que Oracle crea asociados a las claves primarias se borran, además, cuando se elimina dicha restricción, aunque no se elimine la tabla.

Las tablas del catálogo donde se anotan las características de los índices creados son:

- dba_indexes
(index_name, index_type, table_name, uniqueness), que anotan el nombre del índice, su tipo, la tabla sobre la que se ha creado y su condición de unicidad, respectivamente.
- dba_ind_columns
(index_name, table_name, column_name, column_position), que añade a la vista anterior el nombre y la posición de la columna sobre la que se ha montado el índice.

3.4.4. CLÚSTERS

El clúster es un mecanismo que permite indicar al SGBD que el almacenamiento de ciertas tablas ha de hacerse de modo que los datos que contengan se encuentren físicamente cercanos. Por tanto, un clúster lo forman un conjunto de tablas que se

almacenar en los mismos bloques de datos porque comparten campos comunes (clave del clúster) y con frecuencia se accede a ellas de forma conjunta. Esto es, se usan para almacenar en los mismos bloques tablas que comparten atributos y se evita así duplicar valores de columna.

Éste es el caso, por ejemplo, de la definición de claves externas entre tablas o de tablas procedentes de una relación entre una entidad fuerte y las débiles asociadas. Gracias a los clústers se reduce considerablemente el número de operaciones de E/S en las reuniones y tienen como ventaja que no afectan a las aplicaciones ni a los usuarios. Sin embargo, no es aconsejable usar clústers con tablas a las que se accede frecuentemente de forma individual.

El clúster se debe usar para almacenar tablas que son principalmente consultadas y escasamente modificadas. Además, dichas consultas deben reunir varias tablas del clúster. Si no se cumplen estos requisitos, el clúster resultante tendrá un efecto negativo sobre el rendimiento porque no aprovechamos las ventajas que nos ofrece y, sin embargo, estamos pagando el costo de su mantenimiento.

Por ejemplo, las tablas cliente y pedido comparten el campo clienteid (código del cliente). Si se incluyen las dos tablas en el mismo clúster, Oracle almacenará físicamente todas las filas de cada clienteid de ambas tablas en los mismos bloques de datos. Supongamos que la tablas cliente y pedido contienen la siguiente información:

CLIENTEID	NOMBRE	DIREC	CIUDAD
C1	Antonio Martín	C/ Marte	Madrid
C2	David Bailón	C/ Luna	Granada
C3	Lara Croft	C/ Sur	Londres

PEDIDOID	CLIENTEID	FECHA	TOTAL
P1	C2	17/02/02	500
P2	C1	25/07/04	2000
P3	C3	15/06/99	540
P1	C2	27/10/03	2088
P3	C3	25/01/02	4040

Si incluimos ambas tablas en el mismo clúster, compartiendo el campo clienteid, el resultado será el que se muestra en la Tabla 3.1.

TABLA 3.1 Estructura interna del clúster de clientes-pedidos.

CLIENTEID	NOMBRE	DIREC	CIUDAD
C1	Antonio Martín	C/ Marte	Madrid
	PEDIDOID	FECHA	TOTAL
	P2	25/07/04	2000
CLIENTEID	NOMBRE	DIREC	CIUDAD
C2	David Bailón	C/ Luna	Granada
	PEDIDOID	FECHA	TOTAL
	P1	17/02/02	500
	P1	27/10/03	2088
CLIENTEID	NOMBRE	DIREC	CIUDAD
C3	Lara Croft	C/ Sur	Londres
	PEDIDOID	FECHA	TOTAL
	P3	15/06/99	540
	P3	25/01/02	4040

Como los clústers almacenan los registros relacionados en los mismos bloques de datos, los beneficios obtenidos son:

- Se reduce el número de accesos a disco y se mejora el rendimiento de las reuniones que involucran a las tablas del clúster.
 - La clave del clúster (columnas comunes entre las tablas del clúster) almacena cada valor una sola vez, independientemente del número de registros que contengan dicho valor.

Se debe elegir la clave del clúster con cuidado. Si en la consulta que involucra la reunión se igualan varios campos, se debe elegir una clave compuesta para el clúster. Una buena clave debe tener un número adecuado de valores distintos para que la información asociada a cada valor de la clave ocupe aproximadamente un bloque de

datos. Si la cantidad de valores distintos de la clave del clúster es muy pequeña se desperdicia espacio y la mejora en el rendimiento es muy baja.

Para la creación de un clúster se emplea la sentencia CREATE clúster cuya sintaxis básica es:

```
CREATE CLUSTER nombre_del_cluster(campo tipo_de_dato);
```

Por ejemplo, para crear el clúster del ejemplo mostrado anteriormente, usamos la sentencia:

```
CREATE CLUSTER cluster_clientes(clienteid NUMBER(6));
```

Una vez que se ha creado un clúster, se pueden crear las tablas que contendrá. Para ello se emplea la sentencia CREATE TABLE con la opción clúster, para indicar el nombre del clúster al que pertenecerá la tabla y, entre paréntesis, el nombre del campo o campos que componen la clave del mismo.

```
CREATE TABLE cliente (
    clienteid NUMBER(6) PRIMARY KEY,
    nombre     VARCHAR2(45),
    direc     VARCHAR2(40),
    ciudad     VARCHAR2(30)
) cluster cluster-clientes(clienteid);
```

```
CREATE TABLE pedido (
    pedidoid   NUMBER(4) PRIMARY KEY,
    fechaped   DATE,
    clienteid  NUMBER(6) NOT NULL
        REFERENCES cliente(clienteid),
    total      NUMBER(8,2)
) cluster cluster-clientes(clienteid);
```

Antes de que se pueda insertar información en las tablas del clúster es necesario indexar el propio clúster para poder localizar con rapidez el bloque físico donde se ubicará cada registro. Esto se logra mediante la sentencia CREATE INDEX. Por ejemplo, para crear el índice del clúster del ejemplo usaremos:

```
CREATE INDEX indice_cluster ON cluster cluster_clientes;
```

A partir de este instante, se podrán insertar tuplas en las tablas del clúster.

Un clúster puede ser eliminado si las tablas que contiene no son necesarias. Cuando se borra el clúster también se borra el índice asociado al mismo.

Para la eliminación de un clúster se emplea la sentencia **DROP clúster** cuya sintaxis es:

```
DROP cluster nombre_del_cluster  
[ INCLUDING TABLES [CASCADE CONSTRAINTS] ] ;
```

— Si un clúster contiene tablas, no podrá ser eliminado a menos que se eliminen antes las tablas que contiene o que se incluya en la sentencia la opción **INCLUDING TABLES**. El clúster tampoco podrá ser borrado si las tablas que incluye contienen campos que son referenciados por claves externas de tablas ajenas al clúster, salvo que se incluya la cláusula **CASCADE CONSTRAINTS**.

— Se pueden eliminar tablas individuales de un clúster usando la sentencia **DROP TABLE** de la forma habitual. La eliminación de una tabla del clúster no afecta ni al clúster, ni a las otras tablas, ni al índice del clúster.

— El índice de un clúster puede ser eliminado en cualquier momento sin afectar al clúster ni a las tablas del mismo. Sin embargo no será posible acceder a la información contenida en las tablas hasta que éste se reconstruya. Para la eliminación del índice se emplea la sentencia **DROP INDEX**, en su formato habitual.