Otras estructuras de índices. Almacenamiento, índices y clusters en Oracle

1 Otras estructuras de datos de índices en Oracle

1.1 Tablas organizadas en índice (IOT): Índices primarios

Existe un tipo de índices basados en árboles B+ que se denominan tablas organizadas en índice (Index-organized tables, IOT). En este caso, la entrada de datos ***k*** almacena el **registro de datos indexado completo** con valor de clave ***k***. No hay un archivo de índices separado, sino que se trata del propio archivo de datos. Este tipo de índices se crean siempre **sobre la clave primaria de la tabla o un prefijo** de la misma. Es por ello que se denominan **índices primarios** y son el modo de construir índices agrupados en Oracle.

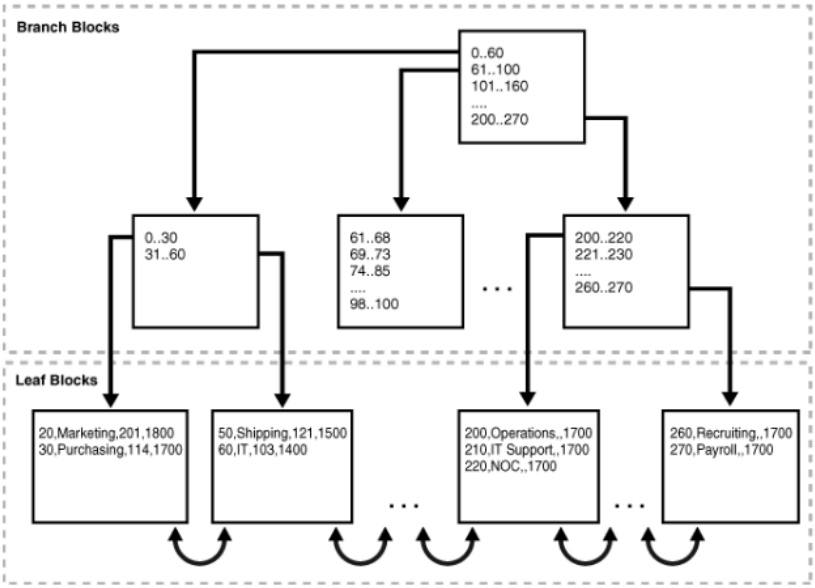


Figura 1.- Ejemplo de tabla organizada en índice (índice primario)

Las tablas organizadas en índice proporcionan un acceso más rápido a las filas de la tabla mediante su clave primaria o un prefijo de la misma. La presencia de la fila completa en el bloque hoja del índice evita una operación de E/S adicional para acceder al bloque de datos. Por ejemplo, el salario del empleado 100 se almacena en la propia fila del índice. Además, debido a que las filas se almacenan en el orden de la clave primaria, el acceso por rango mediante la clave primaria o el prefijo implica un número de operaciones de E/S de bloque mínimas.

Para crear una tabla sólo índice hay que especificar una restricción de clave primaria para la tabla. Sin embargo, el hecho de **crear una restricción de clave primaria no implica que la tabla sea sólo índice**. La tabla sólo índice no tendrá identificadores de fila (RowId) para las filas, ya que se utiliza el valor de la clave primaria como valor de identificador de fila lógico.

Las desventajas de los índices primarios se producen cuando se crea otro índice (denominado **índice secundario**) sobre la misma tabla. Dado que los datos no se almacenan estáticamente como en una tabla aleatoria (heap), pueden tener que ser desplazados en cualquier momento para mantener el orden del índice primario. Por tal motivo, no es posible almacenar en el índice secundario el rowid de las filas dentro de la tabla ordenada según el índice, sino que se debe utilizar una clave lógica.

El siguiente ejemplo muestra una búsqueda sobre un índice para hallar todas las ventas del 23 de mayo de 2012. La Figura 2 muestra el proceso cuando se usa la tabla heap. La ejecución involucra dos etapas: (1) la

búsqueda en el árbol B+; (2) el acceso a los bloques de datos donde se encuentran los registros en la tabla. De este modo, la BD puede cargar de inmediato la fila desde el archivo de datos porque el índice tiene la posición exacta de la página donde se encuentra.

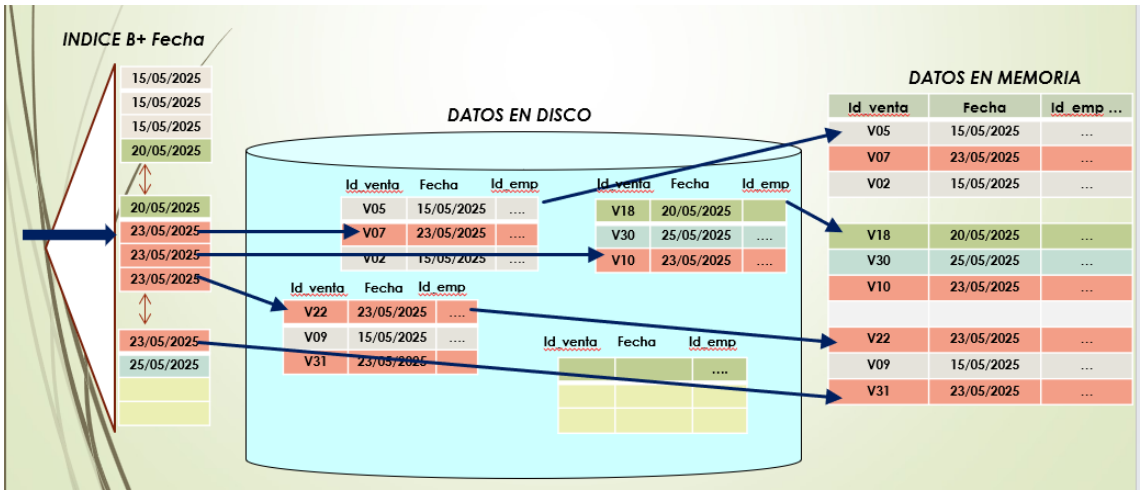


Figura 2.- Acceso basado en índice sobre una tabla heap para localizar las ventas del 23/05/2025

Sin embargo, la situación cambia cuando se utiliza un índice secundario sobre una tabla ordenada según el índice. Tener acceso a una IOT a través de un índice secundario es muy ineficiente. El índice secundario no almacena un puntero físico (RowId) sino una clave lógica para buscar dentro del índice agrupado. Eso significa que tener acceso a la tabla a través de un índice secundario implica buscar sobre 2 índices: una vez sobre el índice secundario, y después sobre el índice agrupado por cada fila encontrada dentro del índice secundario (ver Figura 3).

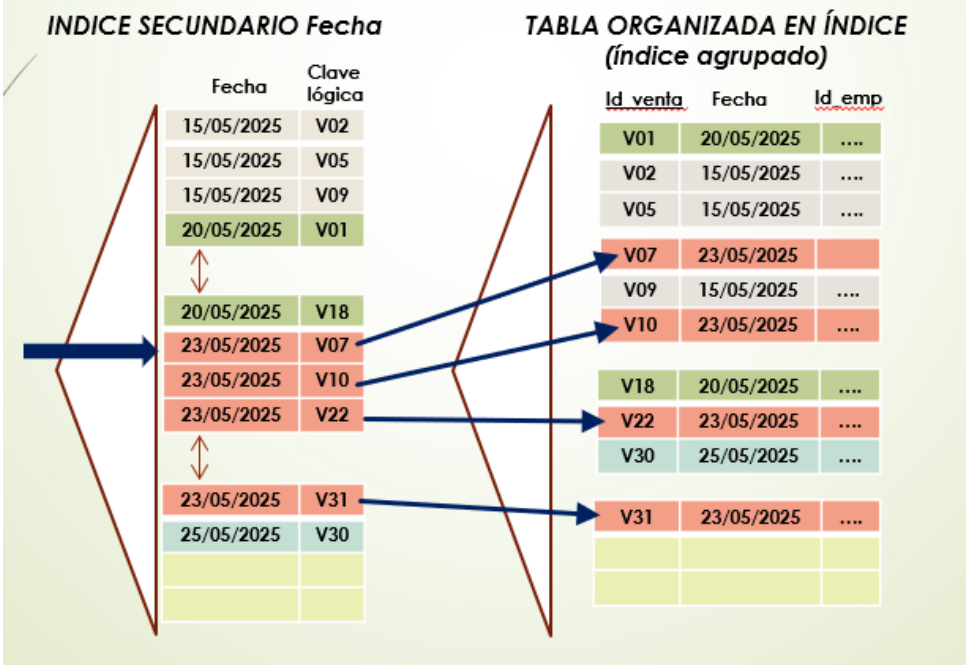


Figura 3.- Índice secundario sobre una IOT

Esta forma de acceso resulta muy ineficiente, por lo que, en general, se recomienda la utilización de índices bajo la **opción sólo-índice** (incluyendo los atributos de la parte SELECT en el índice) en lugar de esta aproximación de tabla ordenada según el índice.

1.2 Índices bitmap (mapas de bits)

Otro tipo de índices útiles para indexar columnas con pocos valores posibles que se repiten a menudo son los índices de **mapas de bits**. La principal ventaja de este tipo de índice es el poco tamaño que ocupa cuando se usa bajo las condiciones adecuadas, permitiendo incluso tener todo el índice cargado en memoria y evitar operaciones de E/S para consultarlo.

Este tipo de índice permite identificar qué filas cumplen una determinada condición utilizando solo operaciones de bits. Como las operaciones sobre secuencias de bits son mucho más rápidas que sus alternativas sobre otros tipos de datos, este índice permite identificar si una fila cumple las condiciones especificadas más rápidamente.

En un índice bitmap, la BD almacena un bitmap para cada valor de clave. A diferencia de un índice B+, donde cada entrada de datos del índice apunta a un único registro de datos, en un índice bitmap, cada entrada de datos del índice almacena rowids a varios registros de datos.

Cada bit en el bitmap se corresponde a un posible rowid de registro. Si el bit está activado, significa que el registro con el rowid correspondiente contiene ese valor clave. Existe una función de mapeo que convierte la posición del bit al rowid actual. De este modo, el índice bitmap proporciona la misma funcionalidad que un índice B+ pero utiliza una representación interna diferente.

EJEMPLO:
```
SELECT ID, LAST_NAME, MARITAL_STATUS, GENDER
FROM   CUSTOMER
ORDER BY id;
```

ID	LAST_NAME	MARITAL_STATUS	GENDER
1	Kessel		M
2	Koch		F
3	Emmerson		M
4	Hardy		M
5	Gowen		M
6	Charles	single	F
7	Ingram	single	F

Un índice bitmap para la columna Gender estaría formado por las siguientes 2 entradas:

Entrada del índice	Rowid1	Rowid 2	Rowid 3	Rowid 4	Rowid 5	Rowid 6	Rowid 7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1

De este modo, la entrada de datos del índice para M se representa con el bitmap 1011100, y la entrada de datos para F se representa con el bitmap 0100011.

La función de mapeo asociada convierte cada bit del bitmap a un rowid de registro en CUSTOMER. El valor de cada bit depende del valor del campo en el registro de datos correspondiente. Por ejemplo, el primer bit del bitmap para el valor M es 1 porque el género para el primer registro de datos es M. Los bits 2, 6 y 7 son 0 porque el género para esos registros de datos no es M.

Este tipo de índices se utiliza cuando:

- El fichero de datos tiene un número elevado de registros de datos. La indexación de un fichero con muchos registros empleando un índice B+ puede ser prohibitivamente caro en términos de espacio, porque el índice podría ser mayor que los datos del fichero. Por el contrario, los índices bitmap son habitualmente una pequeña fracción del tamaño de los datos indexados.
- Las columnas indexadas tienen baja cardinalidad; es decir, el número de valores distintos es pequeño comparado con el número de registros de datos en el fichero. Por ejemplo, el género de los empleados. En general, si el número de valores distintos para la columna indexada es menos de un 1% del número de registros de datos, entonces la columna es candidata para un índice bitmap.

- El fichero indexado es una tabla estática, o no está sujeto a modificaciones significativas. El motivo es que cuando se realiza una modificación sobre una columna indexada se bloquea todo el índice bitmap. Dicho bloqueo impide que se pueda utilizar el índice de nuevo hasta que no se resuelva la transacción que originó el bloqueo. Por lo tanto, si en una tabla de un millón de filas se modificara el valor de una fila se bloquearía todo el índice, provocando que no se pudiera utilizar para acceder a ningún valor de la tabla y ralentizando enormemente otras operaciones que pudieran estar ejecutándose concurrentemente en la tabla.
- Las consultas son por igualdad, especialmente en combinación con los operadores AND, OR y NOT. Los registros de datos que satisfacen las condiciones se filtran antes de acceder al fichero de datos. Esto mejora el tiempo de respuesta, en ocasiones de forma muy elevada.

Ej: Supongamos una tabla con registros como se muestran a continuación:

Fila	CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
1	101	single	east	male	bracket_1
2	102	married	central	female	bracket_4
3	103	married	west	female	bracket_2
4	104	divorced	west	male	bracket_4
5	105	single	central	female	bracket_2
6	106	married	central	female	bracket_3

Supongamos que existen índices bitmap para las columnas MARITAL_STATUS y REGION, que serían de la forma:

MARITAL_STATUS	Rowid1	Rowid 2	Rowid 3	Rowid 4	Rowid 5	Rowid 6
Single	1	0	0	0	1	0
Married	0	1	1	0	0	1
Divorced	0	0	0	1	0	0

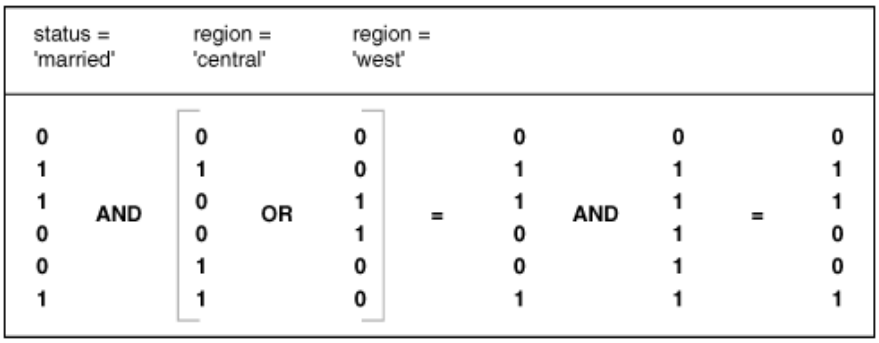
REGION	Rowid1	Rowid 2	Rowid 3	Rowid 4	Rowid 5	Rowid 6
East	1	0	0	0	0	0
Central	0	1	0	0	1	1
West	0	0	1	1	0	0

Entonces, dada la consulta:

```
SELECT COUNT(*) FROM CUSTOMER

WHERE MARITAL_STATUS = 'married' AND REGION IN ('central', 'west');
```

Los índices bitmap pueden procesar esta consulta con gran eficiencia contando el número de 1 en el bitmap resultante, como se muestra a continuación:



Para localizar los clientes que cumplen la condición, bastará con utilizar el bitmap final para acceder a la tabla.

Nota: Las estructuras bitmap no se almacenan persistentemente, sino que se desechan después de la ejecución de la sentencia.

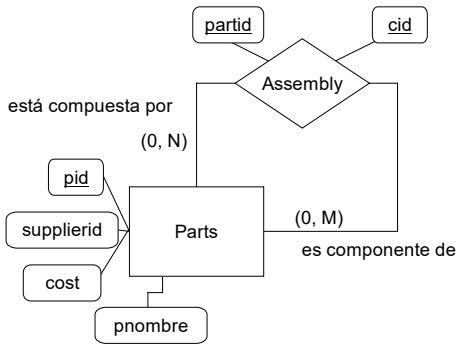
1.3 Agrupación (Clustering) de relaciones

Algunos SGBD (como ORACLE) permiten almacenar los registros de varias relaciones físicamente juntos, en la denominada **agrupación de relaciones** (*clustering*). Las agrupaciones sirven para guardar datos de distintas tablas en los mismos bloques de datos físicos. Se deben utilizar si con frecuencia se consultan de forma conjunta los registros de esas tablas.

Al estar almacenados dichos registros en los mismos bloques (páginas) de datos, se reduce el número de lecturas necesarias para llevar a cabo las consultas y, como consecuencia, se mejora el rendimiento. Sin embargo, estas agrupaciones ejercen un efecto negativo sobre las transacciones de manipulación de datos y sobre las consultas que solo hagan referencia a una de las tablas de la agrupación.

Cada agrupación guarda los datos de las tablas, además de mantener un **índice de clúster** que sirve para ordenar los datos. Las columnas del índice de clúster se denominan ***clave de clúster***, se trata del conjunto de columnas que tienen en común las tablas del clúster. Dado que las columnas de la clave de clúster son las que determinan la ubicación física de las filas en el clúster, estas columnas no deberían actualizarse con mucha frecuencia. La clave de clúster suele ser la clave foránea de una tabla que hace referencia a la clave primaria de otra tabla de clúster.

Veamos su interés bajo el siguiente ejemplo: Supongamos una entidad de piezas (PARTS) y una relación reflexiva ASSEMBLY que asocia cada pieza con sus componentes. Una pieza puede tener muchos componentes, y cada pieza puede ser componente de otras piezas. Su modelo E-R es de la forma:



Supongamos la siguiente consulta que permite conocer los componentes inmediatos de cada pieza:

```
SELECT P.pid, P.pnombre, A.cid
FROM PARTS P, ASSEMBLY A
WHERE P.pid = A.partid
ORDER BY P.pid
```

En este caso, se podría optar por crear los siguientes índices:

Campo	Agrupado /No Agrupado
P.pid, P.pnombre	No Agrupado (se dispone de toda la información en el índice)
A.partid, A.cid	No Agrupado (se dispone de toda la información en el índice)

Otra opción es crear un clúster; es decir, agrupar las dos tablas, almacenando cada registro de PARTS seguido por los registros de ASSEMBLY tales que P.pid=A.partid. Esta aproximación es mejor porque no necesita ningún índice.

Lo mismo sucede si la consulta permitiese localizar los componentes (inmediatos) de todas las piezas de un determinado suministrador X:

SELECT *P.pid, P.pnombre, A.cid*

FROM *PARTS P, ASSEMBLY A*

WHERE *P.pid=A.partid* AND *P.supplierid='X'*

En este caso, podrían crearse los siguientes índices:

Campo	Agrupado /No Agrupado
P.supplerid (si tuplas='X' ↓↓)	No Agrupado (↓↓ duplicados, ordenación cara)
A.partid, A.cid	No Agrupado (se dispone de toda la información en el índice)

De este modo podría aplicarse, en primer lugar, la condición de selección sobre PARTS, traer las tuplas afectadas y luego recuperar el valor *cid* del índice creado para ASSEMBLY en base al valor *P.partid*.

Una opción alternativa sería hacer un join utilizando una organización clustered, que daría lugar a la recuperación del conjunto de tuplas de PARTS y ASSEMBLY (que se almacenan juntas). En este caso, el índice <A.partid, A.cid> no sería necesario, pero se mantendría P.supplerid.

La aproximación mediante clustering es especialmente interesante para moverse en diferentes niveles de la jerarquía pieza-componente. Por ejemplo, una consulta típica es conocer el coste total de una pieza, lo que requiere la necesidad de realizar joins entre PARTS y ASSEMBLY de forma repetitiva. Además, si no se conoce el número de niveles en la jerarquía a priori, ni siquiera se podría expresar esta consulta directamente en SQL dado que el número de joins varía.

En resumen, el clustering:

- puede acelerar los *joins*, en particular los establecidos entre una clave foránea y una principal correspondientes a relaciones 1:N
- la recuperación secuencial de las relaciones es más lenta
- las inserciones, borrados y actualizaciones que alteran las longitudes de los registros son más lentas debido a la sobrecarga producida por el *clustering*.

2 Creación de índices y clústers en Oracle

En Oracle existen tres tipos de índices: índices **de tabla, de clúster** y **de mapa de bits**.

- Un índice *de tabla* almacena los valores de las columnas indexadas de una tabla junto con la ubicación física de la fila, es decir, su RowId.
- Los índices *de clúster* almacenan los valores de clave de clúster (véanse más detalles en el apartado dedicado a Clústers).
- Un índice *de mapa de bits* es un tipo especial de índice de tabla diseñado para dar soporte a consultas de tablas de gran tamaño con columnas que contengan pocos valores distintos para el índice. Resultan especialmente efectivos con datos muy estáticos.

La sentencia CREATE INDEX sirve para crear índices de forma manual y tiene la siguiente sintaxis:

```
CREATE [UNIQUE | BITMAP] INDEX [esquema.]indice
        [ON [esquema.]tabla (atrib1 [ASC|DESC] [,atrib2 [ASC|DESC]]...)]
        [ON CLUSTER [esquema.]cluster]
```

UNIQUE: indica que se trata de un índice único, es decir, no admite más de una tupla con el mismo valor en los atributos que forman el índice. Cuando se crea una tabla con clave primaria, automáticamente se crea un índice único.

ON [esquema.]tabla (atrib1 [ASC|DESC] [,atrib2 [ASC|DESC]]...): especifica la tabla y columna(s) sobre la(s) que se crea el índice.

ON CLUSTER [esquema].cluster: especifica el clúster para el que se crea el índice. Se trata, por tanto, de un *índice de cluster*. En el apartado Clústers se muestra un ejemplo de su utilización.

2.1 Definición de Tablas en Oracle

La determinación del modo de almacenamiento de una tabla en Oracle forma parte de la sintaxis básica de creación de la misma:

```
CREATE TABLE [esquema.]tabla
  (  atrib1 tipo1 [DEFAULT definicion] [NOT NULL] [CHECK ()],
   [,atrib2 tipo2 [DEFAULT definicion] [NOT NULL] [CHECK ()]]...,
   [CONSTRAINT nombre_restr UNIQUE|PRIMARY KEY (atrib1[,atrib2]...)],
   [CONSTRAINT nombre_restr FOREIGN KEY (atrib1 [,atrib2]...)
                REFERENCES [esquema.]tabla(atrib1[,atrib2]...)
                [ON DELETE SET NULL|CASCADE]],
   [CONSTRAINT nombre_restr CHECK condicion])
   [CLUSTER [esquema.]cluster (atrib1[,atrib2]...)]
   [ORGANIZATION HEAP | INDEX ]
```

ORGANIZATION HEAP | INDEX: indica el tipo de organización de la tabla:

- **HEAP**: las filas se almacenan sin ningún orden. Es el valor por defecto.
- **INDEX**: se crea una **tabla ordenada según el índice** (del inglés, **Index-Organized Tables, IOT**). En este caso, toda la tabla se almacena dentro de una estructura de índice B+, con los datos ordenados por la clave primaria de la tabla. Es la forma alternativa de Oracle a la creación de índices agrupados.

2.2 Clústers

Un clúster es un objeto que contiene datos de una o más tablas, cada una de las cuales tiene una o más columnas comunes. Las filas de las tablas que comparten la misma clave de clúster se almacenan juntas. La clave de clúster se indexa utilizando un *índice de clúster* (de tipo árbol-B+), y su valor se almacena una sola vez.

La sintaxis de la sentencia de creación de un clúster es de la forma:

```
CREATE CLUSTER [esquema.]cluster
  (  atrib1 tipo1 [DEFAULT definicion] [NOT NULL],
   [,atrib2 tipo2 [DEFAULT definicion] [NOT NULL]]...)
   [SIZE integer K|M]
   [INDEX sentencia];
```

cluster: especifica el nombre del clúster a crear. Una vez creado se deben añadir las tablas. Un clúster puede contener un máximo de 32 tablas.

atrib1 tipo1 [DEFAULT definicion] [NOT NULL]: especifica el nombre de las columnas que forman la clave de clúster. El número máximo de columnas que pueden formar la clave de clúster son 16. Es necesario que el tipo de datos y el tamaño de las columnas coincidan con las columnas correspondientes en las tablas agrupadas, aunque no es necesario que coincidan sus nombres.

No se pueden especificar restricciones de integridad como parte de la definición de una columna de clave de clúster. Sí es posible, sin embargo, hacerlo en el momento de la creación de las tablas en clúster.

Dentro de los tipos de datos, una columna que forme parte de la clave de clúster no puede ser del tipo LONG, BLOB, CLOB ni BFILE.

SIZE integer K|M: especifica el tamaño en bytes estimado para almacenar el número medio de filas con la misma clave de clúster.

Oracle utiliza la longitud de la clave de clúster para determinar el espacio necesario para las filas que tienen el mismo valor de clave de clúster. Para conocer el tamaño actual se puede consultar el atributo KEY_SIZE de la vista del catálogo USER_CLUSTERS. Si se omite este parámetro, Oracle reserva 1 bloque de datos completo para cada valor de clave de clúster.

INDEX sentencia: Permite definir un índice para el clúster.

Para conocer qué clústeres existen en la BD en un momento determinado pueden consultarse las tablas USER_CLUSTERS, ALL_CLUSTERS del catálogo.

2.3 Ejemplo de creación de clúster

La forma de crear un clúster requiere varios pasos, que veremos con un ejemplo. Supongamos que disponemos de dos entidades: *ticket* y *línea_ticket* entre las cuales existe una relación 1:N, y queremos crear un clúster, de modo que cada ticket esté almacenado físicamente cerca de sus líneas. Los pasos a realizar son los siguientes:

1. Crear el clúster: El clúster es la estructura que permitirá almacenar los tickets con sus líneas. Para ello, debemos indicar la columna que vamos a usar para localizar ambos y ponerlos en el mismo lugar. En este caso, usaremos una columna a la que llamaremos *código_ticket*, que será del mismo tipo y tamaño que ese código tanto en la tabla *ticket* como en *línea_ticket.*

```
CREATE CLUSTER cluster_ticket(
   cod_ticket          NUMBER(5));
```

2. Crear la tabla del lado 1, la tabla *ticket*, e indicar el nombre de la columna que se usará como clave de clúster.

```
CREATE TABLE ticket (
          cod_ticket                    NUMBER(5) NOT NULL,
          cajero                        VARCHAR2(50) NOT NULL,
          fecha                         DATE NOT NULL,
          CONSTRAINT  pk_ticket         PRIMARY KEY (cod_ticket))
          CLUSTER cluster_ticket(cod_ticket);
```

Obsérvese que, en este caso, la columna que identifica el código del ticket no se llama como el clúster, sino *codigo*, pero es del mismo tipo y tamaño.

3. Crear la tabla del lado N, la tabla *línea_ticket*. Además de establecer la clave foránea que referencia a la tabla *ticket*, indicamos que se va a almacenar en el mismo clúster:

```
CREATE TABLE línea_ticket(
          cod_ticket                    NUMBER(5) NOT NULL,
          num_linea                     NUMBER(3) NOT NULL,
          cantidad                      NUMBER(4) NOT NULL,
          codigo_producto               NUMBER(5) NOT NULL,
          precio                        NUMBER(7,2) NOT NULL,
          CONSTRAINT  pk_linea_ticket PRIMARY KEY (cod_ticket, num_linea),
          CONSTRAINT  fk_ticket         FOREIGN KEY (cod_ticket)
                                        REFERENCES ticket(cod_ticket))
          CLUSTER cluster_ticket(cod_ticket);
```

4. Finalmente, se crea el *índice de clúster*. Oracle no crea un índice para el clúster en el momento de la creación. Es necesario crearlo antes de poder ejecutar sentencias DML sobre el mismo. Su sintaxis es de la forma:

```
CREATE INDEX ind_cluster_ticket
   ON CLUSTER cluster_ticket;
```