



Unidad Didáctica 7:

OTROS OBJETOS DE LA BASE DE DATOS

1. Concepto de vista

Una vista es una tabla lógica basada en una tabla u otra vista. Una vista no contienen datos propios, sino que es muy similar a una ventana a través de la cual se pueden visualizar o cambiar datos de tablas. Las tablas en las que se basen las vistas se llaman tablas base. La vista se almacena como una sentencia `SELECT` en el diccionario de datos.

Tabla EMPLOYEES:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALA
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	2400
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	1700
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	1700
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	3100
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600
149	Zlotkey				29-JUL-98	ST_CLERK	2500
174	Abel				24-JAN-00	SA_MAN	10500
176	Taylor				24-MAY-96	SA_REP	11000
176	Taylor				24-MAR-98	SA_REP	8600
176	Taylor				24-MAY-99	SA_REP	7000
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	8300

20 rows selected.


2. ¿Para qué se utilizan las vistas?

- Las vistas restringen el acceso a los datos debido a que pueden mostrar columnas selectivas desde las tablas.
- Las vistas se pueden utilizar para hacer que las consultas sencillas recuperen los resultados de consultas complicadas. Por ejemplo, las vistas se pueden utilizar para consultar información de varias tablas sin necesidad de que el usuario sepa cómo escribir una sentencia de unión.
- Las vistas proporcionan independencia de datos para programas y usuarios. Una vista se puede utilizar para recuperar datos de diversas tablas.

- Las vistas proporcionan a los grupos de usuarios acceso a los datos de acuerdo con sus criterios específicos.

3. Tipos de vistas

Hay dos tipos de vistas: **simples** y **complejas**. La diferencia básica está relacionada con las operaciones DML (**INSERT**, **UPDATE** y **DELETE**).

- Una vista **simple** es aquella que:
 - Deriva datos de sólo una tabla.
 - No contiene funciones ni grupos de datos.
 - Puede realizar operaciones DML a través de la vista.
- Una vista **compleja** es aquella que:
 - Deriva datos de muchas tablas.
 - Contiene funciones o grupos de datos.
 - No siempre permite operaciones DML a través de la vista.


Función	Vistas Simples	Vistas Complejas
Número de tablas	Una	Una o varias
Contiene funciones	No	Sí
Contiene grupos de datos	No	Sí
Operaciones DML a través de una vista	Sí	No siempre

4. Creación de vistas: CREATE VIEW

Podemos crear una vista si embebemos una subconsulta dentro de la sentencia CREATE VIEW.

La sintaxis es:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view [(alias[, alias]...)]  
AS subquery  
[WITH CHECK OPTION [CONSTRAINT constraint]]  
[WITH READ ONLY [CONSTRAINT constraint]];
```



La subconsulta puede contener sintaxis SELECT compleja.

En la sintaxis:

OR REPLACE	vuelve a crear la vista si ésta existe.
FORCE	crea la vista independientemente de si las tablas base existen o no.
NOFORCE	crea la vista sólo si las tablas base existen (es el valor por defecto).
VIEW	es el nombre de la vista.
<i>alias</i>	Lista de alias que se establecen para las columnas devueltas por la consulta SELECT en la que se basa esta vista. El número de alias debe coincidir con el número de columnas devueltas por SELECT .
<i>subquery</i>	es una sentencia SELECT completa.
WITH CHECK OPTION	especifica que sólo las filas accesibles a la vista se pueden insertar o actualizar.
<i>constraint</i>	es el nombre asignado a la restricción CHECK OPTION .
WITH READ ONLY	asegura que no se puede realizar ninguna operación DML en esta vista.

Las vistas no existen realmente como un conjunto de valores almacenados en la base de datos, sino que son tablas ficticias, denominadas *derivadas* (no materializadas). Se construyen a partir de tablas reales (materializadas) almacenadas en la base de datos, y conocidas con el nombre de *tablas básicas* (o tablas de base). La no-existencia real de las vistas hace que puedan ser actualizables o no.

Ejemplo:

Si definimos una vista para saber los clientes que tenemos en Barcelona o en Girona, haríamos:

```
CREATE VIEW clientes_Barcelona_Girona AS
(SELECT *
FROM clientes
WHERE ciudad IN ('Barcelona', 'Girona'))
WITH CHECK OPTION;
```

Si queremos asegurarnos de que se cumpla la condición de la cláusula WHERE, debemos poner la opción WITH CHECK OPTION. Si no lo hiciésemos, podría ocurrir que alguien incluyese en la vista *clientes_Barcelona_Girona* a un cliente nuevo con el código 70, de nombre JMB, con el NIF 36.788.224-C, la dirección en NULL, la ciudad **Lleida** y el teléfono NULL.

Si consultásemos la extensión de la vista *clientes_Barcelona_Girona*, veríamos:

clientes_Barcelona_Girona					
<u>codigo_cli</u>	nombre_cli	nif	direccion	ciudad	telefono
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21

Esta vista sí podría ser actualizable. Podríamos insertar un nuevo cliente con código 50, de nombre CEA, con el NIF 38.226.777-D, con la dirección París 44, la ciudad Barcelona y el teléfono 93.442.60.77.

Después de esta actualización, en la tabla básica clientes encontraríamos, efectivamente:

clientes					
<u>codigo_cli</u>	nombre_cli	nif	direccion	ciudad	telefono
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
50	CEA	38.226.777-D	París, 44	Barcelona	93.442.60.77

Podemos ver la estructura de una vista utilizando el comando `DESCRIBE`.

Cuando creamos una vista debemos tener en cuenta:

- La subconsulta que define una vista puede contener sintaxis `SELECT` compleja, incluyendo uniones, grupos y subconsultas.
- La subconsulta que define la vista no puede contener una cláusula `ORDER BY`. La cláusula `ORDER BY` se especifica cuando se recuperan datos de la vista.
- Si no se especifica un nombre de restricción para una vista creada con `WITH CHECK OPTION`, el sistema asigna un nombre por defecto con el formato `SYS_Cn`.
- Podemos utilizar la opción `OR REPLACE` para cambiar la definición de la vista sin borrarla y volver a crearla .

5. Recuperación de datos de una vista

Podemos recuperar datos de una vista de la misma forma que los recuperamos de una tabla. Podemos visualizar los contenidos de toda la vista o sólo las columnas y filas especificadas.

Ejemplo:

```
SELECT *  
FROM clientes_Barcelona_Girona;
```

Obtendríamos:

clientes_Barcelona_Girona					
<u>codigo_cli</u>	nombre_cli	nif	direccion	ciudad	telefono
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21

6. Modificación de una vista

Con la opción **OR REPLACE** se puede crear una vista incluso si ya existe otra con el mismo nombre, sustituyendo así la versión antigua de la vista para su propietario. Esto significa que la vista se puede modificar sin necesidad de borrar, volver a crear u otorgar, de nuevo, privilegios de objeto.

Ejemplo:

Modificar la vista *clientes_Barcelona_Girona* para que sólo tenga los clientes de estas localidades que no tengan teléfono:

```
CREATE OR REPLACE VIEW clientes_Barcelona_Girona AS
(SELECT *
FROM clientes
WHERE ciudad IN ('Barcelona', 'Girona') AND telefono is NULL)
WITH CHECK OPTION;
```

Si consultamos esta vista, el resultado sería:

clientes_Barcelona_Girona					
<u>codigo_cli</u>	nombre_cli	nif	direccion	ciudad	telefono
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL

7. Borrado de una vista: DROP VIEW

Podemos utilizar la sentencia **DROP VIEW** para eliminar una vista. Esta sentencia elimina la definición de la vista de la base de datos. La eliminación de una vista no tiene ningún efecto en las tablas en las que se basaba la vista. Las vistas u otras aplicaciones basadas en vistas suprimidas se invalidan. Solamente puede eliminar una vista el creador o un usuario con el privilegio **DROP ANY VIEW**.

Ejemplo:

Eliminar la vista *clientes_Barcelona_Girona* :

```
DROP VIEW clientes_Barcelona_Girona;
```

8. Reglas para la realización de operaciones DML en una vista

En las vistas, además de hacer consultas, podemos insertar, modificar y borrar filas. Pero , en cada caso, tenemos que tener en cuenta una serie de reglas:

- Podemos **eliminar una fila** de una vista a no ser que ésta contenga:
 - Funciones de grupo.
 - Una cláusula `GROUP BY`.
 - La palabra clave `DISTINCT`.
 - La palabra clave `ROWNUM` de pseudocolumna.
- Podemos **modificar** los datos de una vista a no ser que ésta contenga:
 - Funciones de grupo.
 - Una cláusula `GROUP BY`.
 - La palabra clave `DISTINCT`.
 - La palabra clave `ROWNUM` de pseudocolumna.
 - Columnas definidas por expresiones (por ejemplo, `SALARIO * 12`).
- Podemos **agregar** datos a través de una vista a no ser que ésta contenga:
 - Funciones de grupo.
 - Una cláusula `GROUP BY`.
 - La palabra clave `DISTINCT`.
 - La palabra clave `ROWNUM` de pseudocolumna.
 - Columnas definidas por expresiones.
 - Columnas `NOT NULL` sin valores por defecto en la tabla base que no estén seleccionadas en la vista. Todos los valores requeridos deben estar presentes en la vista (recordar que se está agregando valores directamente en la tabla subyacente a través de la vista).

Ejemplo:

Creemos una vista que nos dé para cada cliente el número de proyectos que tiene encargados el cliente en cuestión.

```
CREATE VIEW proyectos_por_cliente (codigo_cli, numero_proyectos)
AS
(SELECT c.codigo_cli, COUNT(*)
FROM proyectos p, clientes c
WHERE p.codigo_cliente = c.codigo_cli
GROUP BY c.codigo_cli);
```

El resultado es:

proyectos_por_clientes	
codigo_cli	numero_proyectos
10	2
20	1
30	1

Si alguien insertase en la vista *proyectos_por_cliente*, los valores para un nuevo cliente 60 con tres proyectos encargados, encontraríamos que estos tres proyectos tendrían que figurar realmente en la tabla **proyectos** y, por lo tanto, el SGBD los debería insertar con la información que tenemos, que es prácticamente inexistente. Veamos gráficamente cómo quedarían las tablas después de esta hipotética actualización, que no llegaremos a hacer nunca, ya que iría en contra de la teoría del modelo relacional:

- Tabla **clientes**

clientes					
<u>codigo_cli</u>	nombre_cli	nif	direccion	ciudad	telefono
10	ECIGSA	38.567.893-C	Aragón 11	Barcelona	NULL
20	CME	38.123.898-E	Valencia 22	Girona	972.23.57.21
30	ACME	36.432.127-A	Mallorca 33	Lleida	973.23.45.67
60	NULL	NULL	NULL	NULL	NULL

- Tabla proyectos:

proyectos						
<u>codigo_proyec</u>	nombre_proyec	precio	fecha_inicio	fecha_prev_fin	fecha_fin	codigo_cliente
1	GESCOM	1,0E+6	1-1-98	1-1-99	NULL	10
2	PESCI	2,0E+6	1-10-96	31-3-98	1-5-98	10
3	SALSA	1,0E+6	10-2-98	1-2-99	NULL	20
NULL	NULL	NULL	NULL	NULL	NULL	60
NULL	NULL	NULL	NULL	NULL	NULL	60
NULL	NULL	NULL	NULL	NULL	NULL	60

El SGBD no puede actualizar la tabla básica **clientes** si sólo sabe la clave primaria, y todavía menos la tabla básica **proyectos** sin la clave primaria; por lo tanto, esta vista no sería actualizable.

9. Uso de la cláusula WITH CHECK OPTION

Es posible realizar comprobaciones de integridad referencial a través de las vistas. La vista se puede utilizar para proteger la integridad de datos, pero su uso está muy limitado.

La cláusula **WITH CHECK OPTION** especifica que las inserciones (**INSERT**) y las actualizaciones (**UPDATE**) realizadas a través de la vista no puedan crear filas que la vista no pueda seleccionar y, por tanto, permite que las restricciones de integridad y las comprobaciones de validación de datos se fuercen en los datos que se insertan o se actualizan.

Si hay un intento de realizar operaciones DML en filas que la vista no ha seleccionado, se muestra un mensaje de error junto con el nombre de la restricción, si se ha especificado.

Ejemplo:

```
CREATE OR REPLACE VIEW EMP_20
AS
SELECT *
FROM EMPLE
WHERE DEPT_NO = 20
WITH CHECK OPTION CONSTRAINT EMP20_CK;
```

```
UPDATE EMP_20
SET DEPT_NO=10
WHERE EMP_NO = 7788;
```

Daría error. No se actualiza ninguna fila ya que si el número de departamento tuviera que cambiar a 10, la vista ya no podría mostrar ese empleado. Por lo tanto, con la cláusula

WITH CHECK OPTION, la vista sólo puede mostrar a los empleados del departamento 20 y no permite que el número de departamento de dichos empleados cambie a través de la vista.

10. Denegación de operaciones DML

Podemos asegurarnos que no se produzca ninguna operación DML a agregar la opción **WITH READ ONLY** a la definición de la vista.

Ejemplo:

```
CREATE OR REPLACE VIEW EMP_10 (NUM_EMPLE, APE_EMPLE, CARGO)
AS
SELECT EMP_NO, APELLIDO, OFICIO
FROM EMPLE
WHERE DEPT_NO = 10
WITH READ ONLY;
```

Cuando intento eliminar una fila:

```
DELETE FROM EMP_10
WHERE EMP_NO = 7867;
```

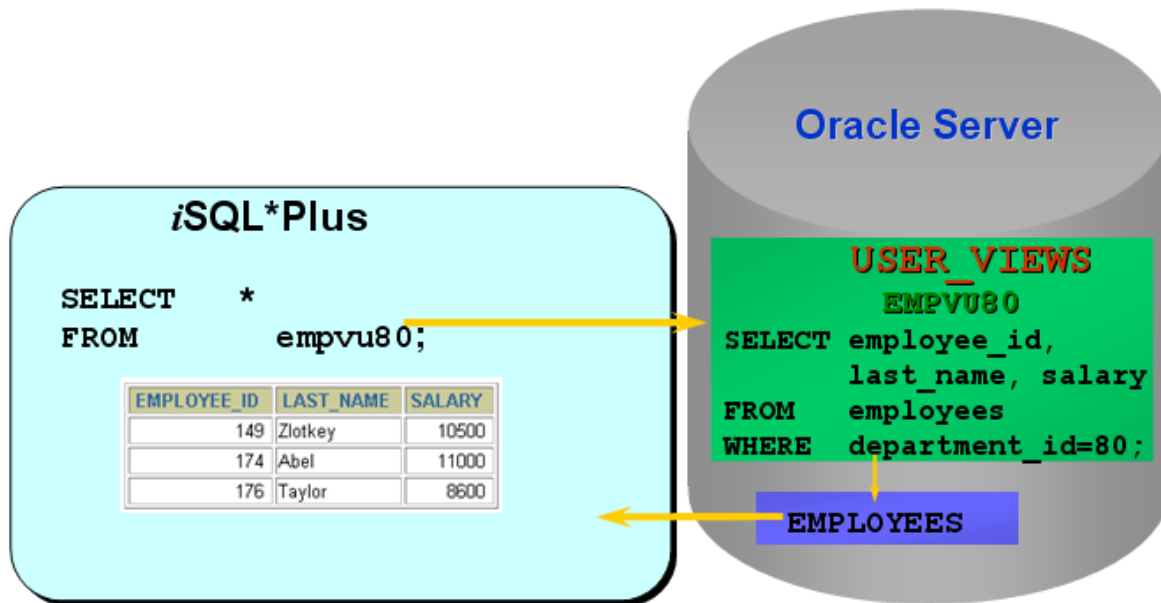
da como resultado un error. De igual forma sucedería con un intento de inserción o modificación.

11. Vistas del diccionario de datos

Una vez que se haya creado la vista, podemos consultar la vista del diccionario de datos llamada **USER_VIEWS** para ver el nombre y la definición de la vista. El texto de la sentencia **SELECT** que constituye la vista se almacena en una columna **LONG**.

Cuando se accede a datos utilizando una vista, Oracle Server realiza las siguientes operaciones:

1. Recupera la definición de la vista de la vista **USER_VIEWS** del diccionario de datos.
2. Comprueba privilegios de acceso para la tabla base de la vista.
3. Convierte la consulta de la vista en una operación equivalente en las tablas o tabla base subyacente. Dicho de otro modo, los datos se recuperan de las tablas base o se realiza una actualización de ellas.



12. Secuencias

Las secuencias (sequences) son objetos que facilitan la generación automática de series numéricas.

Los usos más frecuentes de las secuencias, son:

- La generación automática de claves primarias.
- Coordinar las claves de múltiples filas o tablas.

Las secuencias son independientes de las tablas; por tanto, una misma secuencia se puede usar para generar valores de columnas numéricas de una o más tablas.

Una forma muy común de implementar en las aplicaciones la generación de series numéricas, es forzar que cada transacción que solicita un nuevo número bloquee la tabla, seleccione el último número, lo incremente y libere la tabla. Con esta implementación, si dos usuarios quieren obtener el siguiente número al mismo tiempo, uno de los dos tendrá que esperar a que termine la transacción del otro.

Una de las principales ventajas de las secuencias frente a implementar la generación de éstas series de números en las aplicaciones, es que permiten la generación simultánea de múltiples números garantizando que son únicos. De esta forma, mejora la concurrencia de las aplicaciones.

12.1. Definir secuencias.

Para definir las secuencias, se utiliza el comando CREATE SEQUENCE en el que, además del nombre, se especifican una serie de parámetros: valor inicial, intervalo entre los números, valor mínimo, máximo, etc.

Sintaxis:

```
CREATE SEQUENCE nombre_secuencia
[INCREMENT BY n]
[START WITH n]
[ { MAXVALUE n | NOMAXVALUE } ]
[ { MINVALUE n | NOMINVALUE } ]
[ { CYCLE | NOCYCLE } ]
[ { CACHE n | NOCACHE } ];
```

Ejemplo de secuencia: crear una secuencia llamada DEPT_DEPTNO_SEQ para utilizarla para la PK de la tabla DEPT. No utilizar la opción CYCLE.

```
CREATE SEQUENCE DEPT_DEPTNO_SEQ
INCREMENT BY 10
START WITH 10
MAXVALUE 1000
NOCYCLE;
```

12.2. Pseudocolumnas NEXTVAL y CURRVAL.

Los valores de las secuencias se referencian en comandos SQL usando las pseudocolumnas Nextval y Currval.

Para generar el siguiente número de una secuencia, se referencia a nombre_secuencia.Nextval.

La primera referencia a Nextval devuelve el valor inicial de la secuencia (parámetro START WITH); las sucesivas referencias generan los siguientes números de la serie (parámetro INCREMENT BY).

12.3. Reglas para el uso de NEXTVAL y CURRVAL.

Para referenciar al número actual de una secuencia, se usa nombre_secuencia.Currval. Los usos posibles de Nextval y Currval son:

- Cláusula VALUES de una sentencia INSERT.
- Lista SELECT de una sentencia SELECT que no forme parte de una subconsulta.
- Lista SELECT de una subconsulta en una sentencia INSERT.
- Cláusula SET de una sentencia UPDATE.

No puede utilizarlas en:

- La lista SELECT de una vista.
- La sentencia SELECT con la palabra clave DISTINCT.
- Una sentencia SELECT con cláusula GROUP BY, HAVING u ORDER BY.
- Una subconsulta en una sentencia SELECT, DELETE o UPDATE.
- La expresión DEFAULT en una sentencia CREATE TABLE o ALTER TABLE.

Uso de una secuencia. En el siguiente ejemplo podemos ver la utilidad de las secuencias para insertar un departamento nuevo en la tabla DEPART.

I

```
INSERT INTO DEPART  
VALUES (DEPT_DEPTNO_SEQ.NEXTVAL, 'INFORMATICA', 'VITORIA');
```

13. Índices

Un índice de Oracle Server es un objeto que puede acelerar la recuperación de filas utilizando un puntero. Los índices se pueden crear explícitamente o automáticamente.

Un índice proporciona acceso directo y rápido a las filas de una tabla. Su objeto es reducir la necesidad de E/S de disco utilizando una ruta indexada para encontrar datos rápidamente. Oracle utiliza y mantiene el índice automáticamente. Una vez que se ha creado el índice, no se requiere ninguna actividad directa por parte del usuario.

Los índices son independientes lógicamente y físicamente de su tabla indexada. Esto significa que se pueden crear o borrar en cualquier momento y que no tiene efecto en las tablas base ni en otros índices.

Al borrar una tabla, los índices correspondientes también se borran.

Hay dos tipos de índices:

- **Índices únicos:** Oracle Server lo crea automáticamente al definir una columna en una tabla con una restricción PRIMARY KEY o UNIQUE. El nombre del índice es el nombre que se ha asignado a la restricción.
- **Índices no únicos:** pueden ser manualmente por el usuario. Por ejemplo, podemos crear un índice en una columna FOREIGN KEY para mejorar la velocidad de recuperación.

13.1. Creación de un índice: CREATE INDEX

Podemos crear un índice en una o varias columnas con la sentencia **CREATE INDEX.**

La sintaxis es:

```
CREATE INDEX indice
ON tabla (columna1[, columna2] ...);
```

donde:

indice, es el nombre del índice.

tabla, es el nombre de la tabla.

columna, es el nombre de la columna en la tabla que se va a indexar.

Ejemplo:

Crear un índice sobre la columna apellido de la tabla EMPLE para acelerar la búsqueda de los empleados por sus apellidos.

```
CREATE INDEX ind_ape
ON EMPLE (APELLIDO);
```

13.2. ¿Cuándo se crea un índice?


Más índices en una tabla no significa consultas más rápidas. Cada operación DML que se valida (commit) en una tabla con índices significa que los índices se deben actualizar. Cuantos más índices tenga asociados una tabla, más esfuerzo tiene que realizar Oracle Server para actualizar todos los índices después de una operación DML.

Por lo tanto, se deben crear índices sólo si:

- La columna tiene un amplio rango de valores.
- La columna contiene un gran número de valores nulos.
- Una o más columnas se utilizan juntas frecuentemente en una cláusula WHERE o en una condición de unión.

13.3. ¿Cuándo no se crea un índice?

Normalmente no merece la pena crear un índice si:

- La tabla es pequeña.
- Las columnas no se suelen utilizar como condición en una consulta. 
- La tabla se actualiza frecuentemente.
- Se hace referencia a las columnas indexadas como parte de una expresión.

13.4. Índices en el diccionario de datos

Existen dos vistas en el diccionario de datos que permiten ver la lista de índices creada.

Para confirmar la existencia de los índices existe la vista **USER_INDEXES** del diccionario de datos. La vista **USER_INDEXES** del diccionario de datos contiene el nombre del índice y su unicidad.

La vista **USER_IND_COLUMNS** puede verificar las columnas afectadas por un índice, ya que contiene el nombre del índice, el nombre de la tabla y el nombre de la columna.

13.5. Eliminación de un índice: DROP INDEX

Los índices no se pueden modificar. Para cambiar un índice, debemos borrarlo y volver a crearlo. Podemos eliminar una definición de índice del diccionario de datos emitiendo la sentencia **DROP INDEX.**

Para borrar un índice, debemos ser propietario del mismo o tener privilegio **DROP ANY INDEX.**

Si se borra una tabla, los índices y las restricciones se borran automáticamente, pero las vista y las secuencias se mantienen.

14. Sinónimos

Para consultar una tabla propiedad de otro usuario, es necesario escribir en el nombre de la tabla el nombre del usuario que la creó como prefijo, seguido de un punto. Al crear un sinónimo se elimina la necesidad de cualificar el nombre del objeto con el esquema y se proporciona el nombre alternativo para una tabla, vista, secuencia, procedimiento u otros objetos. Este método puede ser especialmente útil con nombres largos de objeto, o para lograr una referencia más rápida o segura. Los sinónimos permiten abreviaturas o nombres alternativos para los objetos.

Sintaxis:

```
CREATE [PUBLIC] SYNONYM nombre_sinonimo  
FOR objeto;
```



Ejemplo:

Crear un sinónimo público que sea accesible para todos los usuarios llamado DEPT para la tabla DEPARTAMENTOS de Eva:

```
CREATE PUBLIC SYNONYM dept  
FOR eva.departamentos;
```

Para eliminar un sinónimo:

```
DROP SYNONYM nombre_sinonimo;
```

Para consultar los sinónimos en el diccionario de datos tenemos la vista **USER_SYNONYMS** (SYNONYM_NAME, TABLE_OWNER, TABLE_NAME,...).