Egibide

Unidad Didáctica 5:

ELABORACIÓN DEL DISEÑO FÍSICO

1. Introducción al lenguaje SQL

1.1. Características fundamentales

El **lenguaje de consulta estructurado** o **SQL** (por sus siglas en inglés *structured query language*) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en éstas.

El que SQL sea un **lenguaje declarativo** quiere decir que lo importante es definir **qué** se desea hacer, por encima de **cómo** hacerlo (que es la forma de trabajar de los lenguajes de programación de aplicaciones como C o Java).

Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo efectuar consultas con el fin de recuperar -de una forma sencilla- información de interés de una base de datos, así como también hacer cambios sobre ella. Es un lenguaje informático de cuarta generación (4GL).

Se trata de un lenguaje que intenta agrupar todas las funciones que se le pueden pedir a una base de datos, por lo que es el lenguaje utilizado tanto por administradores como por programadores o incluso usuarios avanzados.

1.2. Origenes y evolución

El nacimiento del lenguaje SQL data de 1970 cuando E. F. Codd publica su libro: *"A relational model of data for large shared data banks" (un modelo de datos relacional para grandes bancos de datos compartidos)*". Ese libro dictaría las directrices de las bases de datos relacionales. Apenas dos años después IBM (para quien trabajaba Codd) utiliza las directrices de Codd para crear el *Standard English Query Language* (Lenguaje Estándar Inglés para Consultas) al que se le llamó **SEQUEL**. Más adelante se le asignaron las siglas **SQL** (*Standard Query Language*, lenguaje estándar de consulta) aunque en inglés se siguen pronunciando *secuel*. En español se pronuncia *esecuele*.

En 1979 **Oracle** presenta la primera implementación comercial del lenguaje. Poco después se convertía en un estándar en el mundo de las bases de datos avalado por los organismos ISO y ANSI. En el año 1986 se toma como lenguaje estándar por ANSI de los SGBD relacionales. Un año después lo adopta ISO, lo que convierte a SQL en estándar mundial como lenguaje de bases de datos relacionales.

En 1989 aparece el estándar ISO (y ANSI) llamado SQL89 o SQL1. En 1992 aparece la nueva versión estándar de SQL (a día de hoy sigue siendo la más conocida) llamada SQL92.

En 1999 se aprueba un nuevo SQL estándar que incorpora mejoras que incluyen triggers, procedimientos, funciones,… y otras características de las bases de datos objeto-relacionales; dicho estándar se conoce como SQL99.

El último estándar es el del año 2008 (SQL2008).

Hoy en día todas las bases de datos comerciales cumplen el estándar ANSI, aunque cada fabricante añade sus mejoras al lenguaje SQL.

Año	Nombre	Alias	Comentarios
1986	SQL-86	SQL-87	Primera publicación hecha por ANSI. Confirmada por ISO en 1987.
1989	SQL-89		Revisión menor.
1992	SQL-92	SQL2	Revisión mayor.
1999	SQL:1999	SQL2000	Se agregaron expresiones regulares, consultas recursivas (para relaciones jerárquicas), triggers y algunas características orientadas a objetos.
2003	SQL:2003		Introduce algunas características de XML, cambios en las funciones, estandarización del objeto sequence y de las columnas autonumericas. (Ver Eisenberg et al.: *SQL:2003 Has Been Published*.)
2006	SQL:2006		ISO/IEC 9075-14:2006 Define las maneras en las cuales el SQL se puede utilizar conjuntamente con XML. Define maneras importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma XML. Además, proporciona facilidades que permiten a las aplicaciones integrar dentro de su código SQL el uso de XQuery, lenguaje de consulta XML publicado por el W3C (World Wide Web Consortium) para acceso concurrente a datos ordinarios SQL y documentos XML.
2008	SQL:2008		Permite el uso de la cláusula ORDER BY fuera de las definiciones de los cursores. Incluye los disparadores del tipo INSTEAD OF. Añade la sentencia TRUNCATE. (Ver [1].)

Figura 1. Evolución de SQL

2. Elementos del lenguaje SQL

El código SQL consta de los siguientes elementos:

2.1. Comandos.

Las distintas instrucciones que se pueden realizar desde SQL son:

- **DDL,** *Data Definition Language* (Lenguaje de definición de datos). Permiten modificar la estructura de las tablas de la base de datos. Lo forman las instrucciones CREATE, ALTER, DROP, RENAME y TRUNCATE.

- **DML**, *Data Manipulation Language* (Lenguaje de manipulación de datos). Modifica filas (registros) de la base de datos. Lo forman las instrucciones INSERT, UPDATE y DELETE.

- **DCL**, *Data Control Language* (Lenguaje de control de datos). Administran los derechos y restricciones de los usuarios. Lo forman las instrucciones GRANT y REVOKE.

- **SELECT**. Se trata del comando que permite realizar consultas sobre los datos de la base de datos. Obtiene datos de la base de datos. Es parte del **DML** del lenguaje.

- **Instrucciones de control de transacciones.** Administran las modificaciones creadas por las instrucciones DML. Lo forman las instrucciones ROLLBACK y COMMIT. Se las considera parte del **DML**.

2.2. Cláusulas.

Son palabras especiales que permiten modificar el funcionamiento de un comando (FROM, WHERE, ORDER BY,...).

2.3. Operadores.

Permiten crear expresiones complejas. Pueden ser aritméticos (+,-,*,/,...), de comparación (>, <, !=,<>), lógicos (AND, OR, NOT..).

2.4. Funciones.

Para conseguir valores complejos (SUM(), DATE(),...).

2.5. Literales.

Valores concretos para las consultas: números, textos, caracteres,... Ejemplos: 2, 12.34, 'Avda Cardenal Cisneros'.

2.6. Metadatos.

El diccionario de datos agrupa los metadatos de una base de datos. En este diccionario aparecen todos los objetos de la base de datos; con su nombre, función, control de acceso (seguridad) y correspondencia física en los archivos de datos.

Cada vez que llega al gestor de bases de datos una petición sobre datos de una base de datos, el RDBMS abre el diccionario de datos para comprobar los metadatos relacionados con la petición y resolver si hay permiso de uso y donde localizar físicamente los datos requeridos.

3. Normas de escritura

- En SQL no se distingue entre mayúsculas y minúsculas.
- Las instrucciones finalizan con el signo de punto y coma.
- Cualquier comando SQL (SELECT, INSERT,...) puede ser partidos por espacios o saltos de línea antes de finalizar la instrucción.
- Se pueden tabular líneas para facilitar la lectura si fuera necesario.
- Los comentarios en el código SQL comienzan por /* y terminan por */ (excepto en algunos SGBD).

4. Proceso de las instrucciones SQL

Para poder ejecutar SQL sobre la base de datos, hay que conectar con la instancia Oracle de la base de datos, lo cual requiere la comunicación entre un proceso cliente y el servidor .

Pero de manera general los pasos en ese proceso son:

1. El usuario abre la herramienta que permite el envío de peticiones SQL .
2. El usuario introduce su nombre de usuario y contraseña.
3. Oracle consulta el diccionario de datos para verificar la existencia del usuario y para validar su permiso de conexión. Si lo tiene, se produce la conexión.
4. El usuario escribe la instrucción SQL (supongamos que es una instrucción de modificación).
5. Oracle traduce la instrucción con el analizador de instrucciones (devolvería un error si la instrucción no es válida).
6. Oracle traduce los nombres usados en la instrucción con la ayuda del diccionario de datos.
7. Si es una instrucción de mostrar datos (SELECT), comprueba si otros usuarios han enviado hace poco esa misma instrucción, eso lo comprueba en el caché de instrucciones de la SGA. Si la instrucción está ahí coge los resultados del buffer caché de la base de datos.
8. Si la instrucción conlleva cambios, el servidor bloquea las filas que se modificarán.
9. La base de datos graba los cambios (si los hubo) y actualiza los archivos deshacer.
10. La base de datos graba los nuevos valores para los datos.
11. Oracle libera del bloqueo los registros.
12. El usuario recibe un mensaje de éxito.

5. Descripción de los tipos de datos

A la hora de crear tablas, hay que indicar el tipo de datos de cada campo. Necesitamos pues conocer los distintos tipos de datos. Estos son:

Descripción	Tipos Estándar SQL	Oracle SQL
Texto		
Texto de anchura fija	CHARACTER(*n*) CHAR(*n*)	CHAR(*n*)
Texto de anchura variable	CHARACTER VARYING(*n*) VARCHAR (*n*)	VARCHAR2(*n*)
Texto de anchura fija para caracteres nacionales	NATIONAL CHARACTER(*n*) NATIONAL CHAR(*n*) NCHAR(*n*)	NCHAR(*n*)
Texto de anchura variable para caracteres nacionales	NATIONAL CHARACTER VARYING(*n*) NATIONAL CHAR VARYING(*n*) NCHAR VARYING(*n*)	NVARCHAR2(*n*)

Números		
Enteros pequeños (2 bytes)	SMALLINT	
Enteros normales (4 bytes)	INTEGER INT	
Enteros largos (8 bytes)	BIGINT (en realidad no es estándar, pero es muy utilizado en muchas bases de datos)	
Enteros precisión decimal		NUMBER(*n*)
Decimal de coma variable	FLOAT DOUBLE DOUBLE PRECISSION REAL	NUMBER
Decimal de coma fija	NUMERIC(*m*,*d*) DECIMAL(*m*,*d*)	NUMBER(*m*,*d*)

Fechas		
Fechas	DATE	DATE
Fecha y hora	TIMESTAMP	TIMESTAMP
Intervalos	INTERVAL	INTERVAL

Booleanos y binarios		
Lógicos	BOOLEAN BOOL	
Binarios	BIT BIT VARYING(*n*) VARBIT(*n*)	
Datos de gran tamaño		
Texto gran longitud	CHARACTER LARGE OBJECT CLOB	LONG (en desuso) CLOB
Binario de gran longitud	BINARY LARGE OBJECT BLOB	RAW (en desuso) LONG RAW (en desuso) BLOB

6.1. Textos

Para los textos disponemos de los siguientes tipos (Oracle):

- **VARCHAR .** Para cadenas de caracteres de longitud variable. Su tamaño depende de la base de datos (en Oracle es de 4000). En Oracle se llama **VARCHAR2**, pero es posible seguir utilizando VARCHAR.
- **CHAR.** Para cadenas de caracteres de longitud fija (en Oracle hasta 2000 caracteres).
- **NCHAR.** Para el almacenamiento de caracteres nacionales de texto fijo con posibilidad de cambio de juego de caracteres. Puede almacenar tanto caracteres ASCII, EBCDIC, UNICODE…
- **NVARCHAR.** Para el almacenamiento de caracteres nacionales de longitud variable con posibilidad de cambio de juego de caracteres. Puede almacenar tanto caracteres ASCII, EBCDIC, UNICODE…. En Oracle se llama **NVARCHAR2**.

En todos estos tipos se indican los tamaños entre paréntesis tras el nombre del tipo. Conviene poner suficiente espacio para almacenar los valores. En el caso de los VARCHAR2, no se malgasta espacio por poner más espacio del deseado ya que si el texto es más pequeño que el tamaño indicado, el resto del espacio se ocupa.

6.2. Números

En Oracle, el tipo NUMBER es un formato versátil que permite representar todo tipo de números. Su rango recoge números de entre 10^{-130} y $9{,}99999999999 * 10^{128}$. Fuera de estos rangos Oracle devuelve un error.

Los **números decimales** (números de coma fija) se indican con **NUMBER(*p*,*s*)**, donde *p* es la precisión máxima y *s* es la escala (número de decimales a la derecha de la coma).

Por ejemplo, NUMBER (8,3) indica que se representan números de ocho cifras de precisión y tres decimales. Los decimales en Oracle se presenta con el punto y no con la coma.

Para **números enteros** se indica **NUMBER(*p*)** donde *p* es el número de dígitos. Eso es equivalente a NUMBER(*p*,0).

Para **números de coma flotante** (equivalentes a los float o double de muchos lenguajes de programación) simplemente se indica el texto **NUMBER** sin precisión ni escala.

La cuestión de la precisión y la escala es compleja. Para entenderla mejor, se muestran estos ejemplos:

Formato	Número escrito por el usuario	Se almacena como...
NUMBER	345255.345	345255.345
NUMBER(9)	345255.345	345255
NUMBER(9,2)	345255.345	345255.35
NUMBER(7)	345255.345	Da error de precisión
NUMBER(9,-2)	345255.345	345300
NUMBER(7,2)	345255.345	Da error de precisión

En definitiva, la precisión debe incluir todos los dígitos del número (puede llegar hasta 38 dígitos). La escala sólo indica los decimales que se respetarán del número, pero si es negativa indica ceros a la izquierda del decimal.

6.3. Fechas

- El tipo **DATE** permite almacenar fechas. Las fechas se pueden escribir en formato día, mes y año entre comillas. El separador puede ser una barra de dividir, un guión y casi cualquier símbolo.

  Para almacenar la fecha actual la mayoría de bases de datos proporcionan funciones (como SYSDATE en Oracle) que devuelven ese valor. Las fechas no se pueden manejar directamente, normalmente se usan funciones de conversión. En el caso de Oracle se suele usar **TO_DATE** .

```
Ejemplo:

TO_DATE('3/5/2007')
```

  Internamente un fecha se almacena como el número de días desde cierto punto de inicio (por ejemplo el año 0). Esto permite que las fechas puedan ser tratadas en operaciones aritméticas normales:

```
'1-JAN-2001' + 10 = '11-JAN-2001'
'1-JAN-2000' - 1 = '31-DEC-1999'
'10-MAY-2000' - '1-MAY-2000' = 9
```

- El tipo **TIMESTAMP** es una extensión del anterior. Almacena valores de día, mes y año, junto con hora, minuto y segundos (incluso con decimales). Con lo que representa un instante concreto en el tiempo.

> Un ejemplo de TIMESTAMP sería '2/2/2004 18:34:23,34521'. En este caso si el formato de fecha y hora del sistema está pensado para el idioma español, el separador decimal será la coma (y no el punto).

- Los **intervalos** sirven para almacenar intervalos de tiempo (no fechas, sino una suma de elementos de tiempo). En el caso de Oracle son:

  INTERVAL YEAR TO MONTH

  INTERVAL YEAR TO MONTH almacena un periodo de tiempo utilizando el año y mes de campos de tipo fecha. Use INTERVAL YEAR TO MONTH para representar la diferencia entre dos valores de tipo fecha, donde únicamente las partes significativas son el año y el mes. Por ejemplo, se puede usar este valor para establecer un recordatorio para una fecha 120 meses en el futuro o verificar si han pasado 6 meses desde una fecha en particular.

  Especifique INTERVAL YEAR TO MONTH como sigue:

  ```
  INTERVAL YEAR [(year_precision)] TO MONTH
  ```

  Donde:

  year_precision es el número de dígitos del año. El valor por defecto de year_precision es 2.

```
CREATE TABLE ejemplo1
(duracion INTERVAL YEAR(3) TO MONTH);

INSERT INTO ejemplo1 (duracion)
VALUES (INTERVAL '120' TO MONTH(3));

SELECT TO_CHAR (SYSDATE+ duracion, 'dd-mon-yyyy')
FROM ejemplo1;       -- hoy es 22-Nov-2010

El resultado de esta consulta es:

22-Nov-2020
```

INTERVAL DAY TO SECOND

INTERVAL DAY TO SECOND almacena un periodo de tiempo en términos de días, horas, minutos y segundos. Utilice INTERVAL DAY TO SECOND para representar la diferencia precisa entre dos valores de tipo fecha. Por ejemplo, se puede usar este valor para ser recordado 36 horas después, o para registrar el tiempo entre el inicio y fin de una carrera. Se puede indicar la precisión tras el texto DAY y el número de decimales de los segundos tras el texto SECOND.

Especifique INTERVAL DAY TO SECOND como sigue:

```
INTERVAL DAY [(day_precision)]
TO SECOND [(fractional_seconds_precision)]
```

Donde:

day_precision es el número de dígitos de día en un campo tipo fecha. Los valores aceptados son del 0 al 9. El valor por defecto es 2.

fraccional_seconds_precision es el número de dígitos en la parte fraccional de segundos en el campo de tipo fecha. Los valores aceptados son del 0 al 9. El valor por defecto es 6.

```
CREATE TABLE ejemplo2
(dia_duracion INTERVAL DAY(3) TO SECOND);

INSERT INTO ejemplo2 (dia_duracion)
VALUES (INTERVAL '180' TO DAY(3));

SELECT SYSDATE+ dia_duracion
FROM ejemplo2;       -- hoy es 26-Sep-2010

El resultado de esta consulta es:
25-Mar-2011
```

6.4. Datos de gran tamaño

Son tipos pensados para almacenar datos de tamaño muy grande. No pueden poseer índices ni ser parte de claves.

- **CLOB:**
  Utilizado para almacenar datos de texto de gran tamaño (hasta 4 GB de texto) .
- **BLOB**
  Utilizado para guardar datos binarios de hasta 4 GB de tamaño .

6. DDL

El DDL (Data Definition Language) es la parte del lenguaje SQL que realiza la función de definición de datos del SGBD. Fundamentalmente se encarga de la creación, modificación y eliminación de los objetos de la base de datos (es decir de los **metadatos**). Por supuesto es el encargado de la creación de las tablas.

Cada usuario de una base de datos posee un **esquema**. El esquema suele tener el mismo nombre que el usuario y sirve para almacenar los objetos de esquema, es decir los objetos que posee el usuario.

Esos objetos pueden ser: tablas, vistas, índices y otras objetos relacionados con la definición de la base de datos. Los objetos son manipulados y creados por los usuarios. En principio sólo los administradores y los usuarios propietarios pueden acceder a cada objeto, salvo que se modifiquen los privilegios del objeto para permitir el acceso a otros usuarios.

Objeto	Descripción
Tabla	Unidad básica de almacenamiento; compuesta por filas y columnas
Vista	Lógicamente representa un subconjunto de datos de una o mas tablas
Secuencia	Generador de valores numéricos
Índice	Proporciona desempeño a algunas consultas
Sinónimo	Nombre alternativo proporcionado a los objetos

Hay que tener en cuenta que **ninguna instrucción DDL puede ser anulada por una instrucción ROLLBACK** (la instrucción ROLLBACK está relacionada con el uso de transacciones que se comentarán más adelante) por lo que hay que tener mucha precaución a la hora de utilizarlas. Es decir, las instrucciones DDL generan acciones que no se pueden deshacer (salvo que dispongamos de alguna copia de seguridad).

7.1. Creación de bases de datos

Esta es una tarea administrativa que se comentará más profundamente en otros temas. Por ahora sólo se comenta de forma simple. Crear la base de datos implica indicar los archivos y ubicaciones que se utilizarán para la misma, además de otras indicaciones técnicas y administrativas que no se comentarán en este tema.

Lógicamente sólo es posible crear una base de datos si se tienen privilegios DBA (*DataBase Administrator)* (SYSDBA en el caso de Oracle).

El comando SQL de creación de una base de datos es **CREATE DATABASE**. Este comando crea una base de datos con el nombre que se indique. Ejemplo:

```
CREATE DATABASE prueba;
```

Pero normalmente se indican más parámetros. Ejemplo (parámetros de Oracle):

```
CREATE DATABASE prueba
LOGFILE prueba.log
MAXLOGFILES 25
MAXINSTANCES 10
ARCHIVELOG
CHARACTER SET WIN1214
NATIONAL CHARACTER SET UTF8
DATAFILE prueba1.dbf AUTOEXTEND ON MAXSIZE 500MB;
```

7.2. Creación de tablas

Nombre de las tablas

Deben cumplir las siguientes reglas (reglas de Oracle, en otros SGBD podrían cambiar):

- Deben comenzar con una letra.
- No deben tener más de 30 caracteres.
- Sólo se permiten utilizar letras del alfabeto (inglés), números o el signo de subrayado (también el signo $ y #, pero esos se utilizan de manera especial por lo que no son recomendados).
- No puede haber dos tablas con el mismo nombre para el mismo esquema (pueden coincidir los nombres si están en distintos esquemas).
- No puede coincidir con el nombre de una palabra reservada SQL (por ejemplo no se puede llamar SELECT a una tabla).
- En el caso de que el nombre tenga espacios en blanco o caracteres nacionales (permitido sólo en algunas bases de datos), entonces se suele entrecomillar con comillas dobles. En el estándar SQL 99 (respetado por Oracle) se pueden utilizar comillas dobles al poner el nombre de la tabla a fin de hacerla sensible a las mayúsculas (se diferenciará entre “FACTURAS” y “Facturas”).

Orden CREATE TABLE

Es la orden SQL que permite crear una tabla. Por defecto será almacenada en el espacio y esquema del usuario que crea la tabla. Una sintaxis general sería:

```
CREATE TABLE <nombre de tabla>
(nombre_columna1 tipo [restricción de columna],
........
nombre_columnaN tipo [restricción de columna],
[restricción_de_tabla]);
```

Una sintaxis más específica es:

```
CREATE TABLE <nombre tabla>
( <nombre columna> <tipo de dato>
[NOT NULL] [UNIQUE] [CONSTRAINT <nombre restricción>]
[PRIMARY KEY] [REFERENCES][DEFAULT][CHECK]
| [PRIMARY KEY (<lista columnas>)]
| [FOREIGN KEY (<lista columnas>) REFERENCES (nombretabla)]
| [UNIQUE (<lista columnas>)]
  [CONSTRAINT <nombre restricción>],[,...] )
| [CHECK (condición de búsqueda)]);
```

Ejemplo: Creación de la tabla FACTURA de clientes.

```
CREATE TABLE FACTURA
(
   REFERENCIA VARCHAR2(10) NOT NULL,
   DESCRIPCION VARCHAR2(50),
   C_PAIS NUMBER(3),
   C_CLIENTE NUMBER(5),
   IMPORTE NUMBER(12),
   CONSTRAINT PK_FACTURA PRIMARY KEY( REFERENCIA )
   CONSTRAINT FK_CLIENTE(C_CLIENTE)
         REFERENCES CLIENTE(C_CLIENTE) ON DELETE CASCADE
);
```

Sólo se podrá crear la tabla si el usuario posee los permisos necesarios para ello. Si la tabla pertenece a otro esquema (suponiendo que el usuario tenga permiso para grabar tablas en ese otro esquema), se antepone al nombre de la tabla , el nombre del esquema:

```
CREATE TABLE [esquema.] nombreDeTabla
(nombreDeLaColumna1 tipoDeDatos [, ...]);
```

```
Ejemplo: Crear la tabla FACTURA en el esquema Eva.

CREATE TABLE EVA.FACTURA
(
   REFERENCIA VARCHAR2(10) NOT NULL,
   DESCRIPCION VARCHAR2(50),
   C_PAIS NUMBER(3),
   C_CLIENTE NUMBER(5),
   IMPORTE NUMBER(12),
   CONSTRAINT PK_FACTURA PRIMARY KEY( REFERENCIA )
   CONSTRAINT FK_CLIENTE(C_CLIENTE)
         REFERENCES CLIENTE(C_CLIENTE) ON DELETE CASCADE
);
```

Listar estructura de las tablas

Para obtener la estructura (descripción de una tabla) el mandato que se debe emplear es:

```
SQL>DESCRIBE nombre_tabla;
```

```
DESCRIBE CLIENTES;
```

Name	Null?	Type
NUMERO	NOT NULL	NUMBER(38)
FECHA	NOT NULL	DATE
NOMBRE	NOT NULL	CHAR(30)
TELEFONO	NOT NULL	CHAR(20)
DIRECCION		CHAR(100)

Opción DEFAULT

Una columna puede tomar un valor por defecto con el uso de la opción DEFAULT. Esta opción previene valores nulos para columnas si una fila es insertada sin valores para dichas columnas.

El valor por defecto puede ser un literal, una expresión o una función SQL, como SYSDATE, pero el valor no puede ser el nombre de otra columna o una pseudocolumna. La expresión por defecto debe corresponder al tipo de dato de la columna.

Se puede poner esta propiedad durante la creación o modificación de la tabla, añadiendo la palabra DEFAULT tras el tipo de datos del campo y colocando detrás el valor que se desea por defecto.

```
Ejemplo:

CREATE TABLE articulo (
cod NUMBER(7), nombre VARCHAR2(25),
precio NUMBER(11,2) DEFAULT 3.5);
```

Restricciones

Una restricción es una condición de obligado cumplimiento para una o más columnas de la tabla.

Su sintaxis general es:

```
CREATE TABLE nombreTabla |
ALTER TABLE nombreTabla {ADD | MODIFY}}
(campo tipoDeDatos [propiedades]
[[CONSTRAINT nombreRestricción ]] tipoRestricción (columnas)
[,siguienteCampo…]
[,CONSTRAINT nombreRestricción tipoRestricción (columnas) ...)
```

Las restricciones tienen un nombre. En el caso de no ponerle nosotros un nombre (algo poco recomendable) entonces el propio Oracle le coloca el nombre que es un mnemotécnico con el nombre de tabla, columna y tipo de restricción (con el nombre *SYS_C00n*, donde "n" es un número asignado automáticamente por Oracle).

Los nombres de restricción no se pueden repetir para el mismo esquema, debemos de buscar nombres únicos. Es buena idea incluir de algún modo el nombre de la tabla, los campos involucrados y el tipo de restricción en el nombre de la misma.

Desde la empresa Oracle se aconseja la siguiente regla a la hora de poner nombre a las restricciones:

- Tres letras para el nombre de la tabla.
- Carácter de subrayado.
- Tres letras con la columna afectada por la restricción.
- Carácter de subrayado.
- Dos letras con la abreviatura del tipo de restricción. La abreviatura puede ser:

  NN. NOT NULL.

  PK. PRIMARY KEY

  UK. UNIQUE

  FK. FOREIGN KEY

  CK. CHECK (validación)

Por ejemplo para hacer que la clave principal de la tabla *Alumnos* sea el *cód_alumno*, el nombre de la restricción podría ser:

```
alu_cod_pk
```

Prohibir nulos: NOT NULL

La restricción NOT NULL permite prohibir los nulos en una determinada tabla. Eso obliga a que la columna tenga que tener obligatoriamente un valor para que sea almacenado el registro.

Se puede colocar durante la creación (o modificación) del campo añadiendo la palabra NOT NULL tras el tipo:

```
Ejemplo: Crear la tabla CLIENTE con la restricción de que el
DNI sea obligatorio:

CREATE TABLE cliente(dni VARCHAR2(9) NOT NULL);

En ese caso el nombre le coloca la propia base de datos (en el
caso de Oracle el nombre sería algo como SY002341 por
ejemplo).

Para poner el nombre se usa:

CREATE TABLE cliente(
dni VARCHAR2(9) CONSTRAINT cli_dni_nn NOT NULL
);
```

La restricción NOT NULL es la única que sólo se puede poner seguida al nombre de la columna a la que se aplica (la razón es que NOT NULL sólo se puede aplicar a un campo a la vez).

Valores unicos: UNIQUE

Las restricciones de tipo UNIQUE obligan a que el contenido de una o más columnas no puedan repetir valores. Nuevamente hay dos formas de colocar esta restricción:

```
CREATE TABLE cliente(dni VARCHAR2(9) UNIQUE);

En ese caso el nombre de la restricción la coloca el sistema.
Otra forma es:

CREATE TABLE cliente(
dni VARCHAR2(9) CONSTRAINT cli_dni_uk UNIQUE);

Esta forma permite poner un nombre a la restricción.
```

Si la repetición de valores se refiere a varios campos, la forma sería:

```
CREATE TABLE alquiler
(dni VARCHAR2(9),
 cod_pelicula NUMBER(5),
 CONSTRAINT alquiler_uk UNIQUE(dni,cod_pelicula) ;
```

La coma tras la definición del campo cod_pelicula hace que la restricción sea independiente de ese campo. Eso obliga a que, tras UNIQUE se indique la lista de campos.

Incluso para un solo campo se puede colocar la restricción al final de la lista en lugar de definirlo a continuación del nombre y tipo de la columna.

Las claves candidatas deben llevar restricciones UNIQUE y NOT NULL.

Clave primaria: PRIMARY KEY

La clave primaria de una tabla la forman las columnas que identifican a cada registro de la misma. La clave primaria hace que los campos que la forman sean NOT NULL (sin posibilidad de quedar vacíos) y que los valores de los campos sean de tipo UNIQUE (sin posibilidad de repetición).

Si la clave está formada por un solo campo basta con:

```
CREATE TABLE cliente(
dni VARCHAR(9) PRIMARY KEY,
nombre VARCHAR(50)) ;
```

O, poniendo un nombre a la restricción:

```
CREATE TABLE cliente(
dni VARCHAR(9) CONSTRAINT CLI_DNI_PK PRIMARY KEY,
nombre VARCHAR(50)) ;
```

Si la clave está formada por más de un campo:

```
CREATE TABLE alquiler(dni VARCHAR(9),
cod_pelicula NUMBER(5),
CONSTRAINT ALQ_PK PRIMARY KEY(dni,cod_pelicula)) ;
```

Además:

- Debe ser única, no nula y obligatoria.
- Existe una como máximo por tabla.
- Puede ser referenciada por una clave ajena.
- Crea un índice, que facilita el acceso a la tabla, automáticamente cuando se habilita o se crea.
- Se comporta como única y obligatoria sin necesidad de indicarlo explícitamente.

Restricciones de validación: CHECK

Son restricciones que dictan una condición que deben cumplir los contenidos de una columna. Una misma columna puede tener múltiples CHECKS en su definición (se pondrían varios CONSTRAINT seguidos, sin comas).

Ejemplo:

```
CREATE TABLE ingresos(
cod NUMBER(5) PRIMARY KEY,
concepto VARCHAR2(40) NOT NULL,
importe NUMBER(11,2) CONSTRAINT importe_min CHECK (importe>0)
CONSTRAINT importe_max CHECK (importe<8000) );
```

En este caso la CHECK prohíbe añadir datos cuyo importe no esté entre 0 y 8000.

Para poder hacer referencia a otras columnas hay que construir la restricción de forma independiente a la columna (es decir al final de la tabla):

```
CREATE TABLE ingresos(
cod NUMBER(5) PRIMARY KEY,
concepto VARCHAR2(40) NOT NULL,
importe_max NUMBER(11,2),
importe NUMBER(11,2),
CONSTRAINT importe_maximo CHECK (importe<importe_max)
);
```

Clave foránea: FOREIGN KEY

Una clave secundaria o foránea, es uno o más campos de una tabla que están relacionados con la clave principal (o incluso con una clave candidata) de otra tabla.

La forma de indicar una clave foránea (aplicando una restricción de integridad referencial) es:

```
CREATE TABLE alquiler(
dni VARCHAR2(9) CONSTRAINT alquiler_dni_fk REFERENCES clientes(dni),
cod_pelicula NUMBER(5) CONSTRAINT alquiler_pelicula_fk REFERENCES
peliculas(cod),
CONSTRAINT alquiler_pk PRIMARY KEY(dni,cod_pelicula)
);
```

Significa esta instrucción (en cuanto a claves foráneas) que el campo *dni* se relaciona con la columna *dni* de la tabla clientes.

Si el campo al que se hace referencia es la clave principal, se puede obviar el nombre del campo:

```
CREATE TABLE alquiler(
dni VARCHAR2(9) CONSTRAINT alquiler_dni_fk REFERENCES clientes,
cod_pelicula NUMBER(5) CONSTRAINT alquiler_pelicula_fk REFERENCES
peliculas,
CONSTRAINT alquiler_pk PRIMARY KEY(dni,cod_pelicula) );
```

En este caso se entiende que los campos hacen referencia a las claves principales de las tablas referenciadas (si la clave está formada por más de un campo, el orden de los campos debe de ser el mismo).

Esto forma una relación entre dichas tablas, que además obliga al cumplimiento de la **integridad referencial**. Esta integridad obliga a que cualquier *dni* incluido en la tabla `alquiler` tenga que estar obligatoriamente en la tabla de `clientes`. De no ser así el registro no será insertado en la tabla (ocurrirá un error).

Otra forma de crear claves foráneas (útil para claves formadas por más de un campo) es:

```
CREATE TABLE existencias(
tipo CHAR2(9),
modelo NUMBER(3),
n_almacen NUMBER(1)
cantidad NUMBER(7),
CONSTRAINT exi_t_m_fk FOREIGN KEY(tipo,modelo) REFERENCES piezas,
CONSTRAINT exi_nal_fk FOREIGN KEY(n_almacen) REFERENCES almacenes,
CONSTRAINT exi_pk PRIMARY KEY(tipo,modelo,n_almacen)
);
```

Si la definición de clave secundaria se pone al final hace falta colocar el texto **FOREIGN KEY** para indicar en qué campos se coloca la restricción de clave foránea. En el ejemplo anterior es absolutamente necesario que la clave principal de la tabla `piezas` a la que hace referencia la clave la formen las columnas *tipo* y *modelo* y en que estén en ese orden.

La **integridad referencial** es una herramienta imprescindible de las bases de datos relacionales. Pero provoca varios problemas. Por ejemplo, si borramos un registro en la tabla principal que está relacionado con uno o varios de la secundaria ocurrirá un error, ya que de permitírsenos borrar el registro ocurrirá fallo de integridad (habrá claves secundarias refiriéndose a una clave principal que ya no existe).

Por ello se nos pueden ofrecer soluciones a añadir tras la cláusula **REFERENCES**. Son:

- **ON DELETE SET NULL**. Coloca nulos todas las claves secundarias relacionadas con la borrada.
- **ON DELETE CASCADE**. Borra todos los registros cuya clave secundaria es igual que la clave del registro borrado.
- **ON DELETE SET DEFAULT**. Coloca en el registro relacionado el valor por defecto en la columna relacionada
- **ON DELETE NOTHING**. No hace nada.

En el caso explicado se aplicarían las cláusulas cuando se eliminen filas de la clave principal relacionada con la clave secundaria. En esas cuatro cláusulas se podría sustituir la palabra DELETE por la palabra UPDATE, haciendo que el funcionamiento se refiera a cuando se modifica un registro de la tabla principal; en muchas bases de datos se admite el uso tanto de ON DELETE como de **ON UPDATE**.

En la base de datos Oracle sólo se permite utilizar ON DELETE SET NULL u ON DELETE CASCADE. No se admite el uso de ON UPDATE en ningún caso.

La sintaxis completa para añadir claves foráneas es:

```
CREATE TABLE tabla(lista_de_campos
CONSTRAINT nombreRestriccion FOREIGN KEY (listaCampos)
REFERENCES tabla(clavePrincipalRelacionada)
[ON DELETE | ON UPDATE
[SET NULL | CASCADE | DEFAULT]
);
```

Si es de un solo campo existe esta alternativa:

```
CREATE TABLE tabla(lista_de_campos tipos propiedades,
nombreCampoClaveSecundaria
CONSTRAINT nombreRestriccion
REFERENCES tabla(clavePrincipalRelacionada)
[ON DELETE | ON UPDATE
[SET NULL | CASCADE | DEFAULT]
);
```

Ejemplo:

```
CREATE TABLE alquiler(
dni VARCHAR(9),
cod_pelicula NUMBER(5),
CONSTRAINT alquiler_pk PRIMARY KEY(dni,cod_pelicula),
CONSTRAINT alquiler_dni_fk FOREIGN KEY (dni) REFERENCES clientes(dni)
ON DELETE SET NULL,
CONSTRAINT alquiler_pelicula_fk FOREIGN KEY (cod_pelicula)
REFERENCES peliculas(cod) ON DELETE CASCADE
);
```

Creando una tabla a partir de las filas de otra tabla

Un segundo método para la creación de una tabla es aplicar las cláusula **AS subquery**, el cual crea la tabla e inserta las filas obtenidas por la subconsulta.

Sintaxis:

```
CREATE TABLE tabla [columna1, columna2,...)]
AS subconsulta;
```

Donde:

tabla es el nombre de la tabla.

columna es el nombre de la columna, el valor por defecto y las reglas de integridad.

subconsulta es la sentencia SELECT que define el conjunto de filas que serán insertadas en la nueva tabla.

Normas a seguir:

- La tabla es creada con los nombres de columnas especificados y las filas recuperadas por la sentencia SELECT son insertadas en la tabla.
- La definición de columnas puede contener solo nombre de columnas y valores por defecto.
- Si las especificaciones de la columna se dan, el número de columnas debe ser igual al número de columnas especificadas en la lista SELECT de la subconsulta.
- Si las especificaciones de la columna no se dan, el nombre de la columna en la tabla será igual que el nombre de la columna en la subconsulta.
- Las reglas de integridad no son pasadas a la nueva tabla, únicamente son pasadas las definiciones del tipo de dato.

```
CREATE TABLE dept_80
AS
SELECT emp_no, apellido, salario * 12 SALARIO_ANUAL, fecha_nac
FROM emple
WHERE dept_no = 80;
```

En el ejemplo anterior se crea una tabla llamada DEPT80, que contiene los detalles de todos los empleados trabajando en el departamento 80. Note que los datos para la tabla DEPT80 provienen de la tabla EMPLE.

Asegúrese de proporcionar un alias de columna cuando seleccione una expresión. A la expresión SALARIO * 12 se le proporciona el alias SALARIO_ANUAL.

7.3. Eliminar una tabla

La sentencia **DROP TABLE** seguida del nombre de una tabla permite eliminar la tabla en cuestión. Cuando se elimina una tabla, la base de datos pierde todos los datos de la tabla. Cualquier vista y sinónimo referente a la tabla seguirá existiendo, pero ya no funcionarán (conviene eliminarlos).

Sintaxis:

```
DROP TABLE table;
```

Donde:

table es el nombre de la tabla.

Reglas:

- Todos los datos son borrados de la tabla.
- Cualquier vista y sinónimo que haga referencia a la tabla, permanece pero son inválidos.
- Cualquier transacción pendiente es aceptada (COMMIT).
- Solo el creador de la tabla o un usuario con privilegios DROP ANY TABLE puede eliminar la tabla.

Nota: La sentencia DROP TABLE, una vez ejecutada, no se puede revertir su resultado. El servidor de Oracle no cuestiona la acción cuando se utiliza la sentencia DROP TABLE. Si eres dueño de la tabla o tienes un nivel alto de privilegios, entonces la tabla es inmediatamente eliminada. Como con todas las sentencias DDL, DROP TABLE es cometida automáticamente.

Un caso especial se presenta si la tabla a ser eliminada está referenciada por claves foráneas en otra(s) tabla(s). En este caso, se debe hacer la llamada de la siguiente manera:

```
DROP TABLE nombretabla CASCADE CONSTRAINTS;
```

De esta manera se eliminarán todas las restricciones de clave foránea en otras tablas que referencien a la clave primaria de la tabla a ser eliminada. Si no se incluye el parámetro CASCADE CONSTRAINTS y existe alguna referencia a una tupla que se eliminará, ORACLE retornará un mensaje de error y no eliminará la relación.

7.4. Modificar la estructura de una tabla

Después que ha creado una tabla, puede necesitar cambiar la estructura de la tabla puesto que: ha omitido una columna, la definición de alguna columna necesita ser cambiada, o necesitas eliminar columnas. Todo lo anterior se puede realizar por medio de la sentencia **ALTER TABLE**.

Se puede añadir, modificar y borrar columnas de una tabla utilizando la sentencia ALTER TABLE.

Además, la orden ALTER TABLE permite añadir o eliminar restricciones en una tabla (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK).

Añadir columnas a una tabla

Sintaxis:

```
ALTER TABLE nombreTabla ADD(
nombreColumna1    tipoDatos    [propiedades]
[,nombrecolumna2  tipoDatos    [propiedades]...);
```

Permite añadir nuevas columnas a la tabla. Se deben indicar su tipo de datos y sus propiedades si es necesario.

Las nuevas columnas se añaden al final, no se puede indicar otra posición (hay que recordar que el orden de las columnas no importa).

Ejemplo:

```
ALTER TABLE facturas ADD (fecha DATE);
```

Nota: si una tabla contiene filas cuando una columna es agregada, entonces la nueva columna es iniciada con un valor nulo para todas las filas.

Ejemplo:

```
ALTER TABLE emple ADD (dept_no varchar2(3) NOT NULL);
```

En este caso, para poder indicar la restricción NOT NULL la tabla debe estar vacía. Si la tabla contiene filas, Oracle generaría un error. Una solución es:

```
ALTER TABLE emple ADD (dept_no varchar2(3) DEFAULT '118');
```

Se crea la nueva columna con un valor por defecto: el empleado pertenece al departamento 118.

Borrar columnas en una tabla

Sintaxis:

```
ALTER TABLE nombreTabla DROP(columna1 [,columna2,...]);
```

Elimina la columna indicada de manera irreversible e incluyendo los datos que contenía. No se puede eliminar la única columna de una tabla que sólo tiene esa columna (habrá que usar DROP TABLE).

```
ALTER TABLE facturas DROP (fecha);
```

Al igual que en el caso anterior, en SQL estándar se puede escribir el texto COLUMN tras la palabra DROP.

En el caso en que la columna a eliminar esté siendo referenciada por una clave foránea en otra tabla podemos añadir la clausula **CASCADE CONSTRAINTS** para eliminar la restricción en la otra tabla.

Por ejemplo en:

```
CREATE TABLE curso(
cod_curso CHAR(7) PRIMARY KEY,
fecha_inicio DATE,
fecha_fin DATE,
titulo VARCHAR2(60),
cod_siguientecurso CHAR(7),
CONSTRAINT fecha_ck CHECK(fecha_fin>fecha_inicio),
CONSTRAINT cod_ste_fk FOREIGN KEY(cod_siguientecurso)
REFERENCES curso ON DELETE SET NULL);
```

Produce error esta instrucción:

```
ALTER TABLE curso DROP(fecha_inicio);
```

ERROR en línea 1: ORA-12991: se hace referencia a la columna en una restricción de multicolumna.

El error se debe a que no es posible borrar una columna que forma parte de la definición de una restricción. La solución es utilizar **CASCADE CONSTRAINTS** pues elimina las restricciones en las que la columna a borrar estaba implicada:

```
ALTER TABLE curso DROP(fecha_inicio) CASCADE CONSTRAINTS;
```

Esta instrucción elimina la restricción de tipo CHECK en la que aparecía la fecha_inicio y así se puede eliminar la columna.

Modificar columnas

Permite cambiar el tipo de datos y propiedades de una determinada columna.

Sintaxis:

```
ALTER TABLE nombreTabla MODIFY(columna1 tipo [propiedades]
[columna2 tipo [propiedades] …]);
```

Las modificaciones a las columnas pueden incluir cambios al tipo de dato, tamaño y valor por defecto.

Los cambios que se permiten son (en Oracle):

- Se puede incrementar el tamaño o precisión de una columna numérica.
- Se puede incrementar el tamaño de columnas numéricas o carácter.
- Sólo se puede reducir la anchura de un campo si esa columna posee nulos en todas las filas, o todos los valores son tan pequeños como la nueva anchura o si la tabla no contiene filas.
- Se puede convertir una columna con el tipo de dato CHAR a VARCHAR2 o convertir una columna VARCHAR2 a CHAR solo si las columnas contienen valores nulos o si no cambia su tamaño.
- El cambio del valor por defecto de una columna afecta únicamente a las subsecuentes inserciones a la tabla.
- Se puede pasar de DATE a TIMESTAMP y viceversa.
- Cualquier otro cambio sólo es posible si la tabla está vacía.

Ejemplo:

```
ALTER TABLE facturas MODIFY(fecha TIMESTAMP NOT NULL);
```

Renombrar columnas

Esto permite cambiar el nombre de una columna.

Sintaxis

```
ALTER TABLE nombreTabla
RENAME COLUMN nombreAntiguo TO nombreNuevo;
```

Ejemplo:

```
ALTER TABLE facturas RENAME COLUMN fecha TO fechaYhora;
```

7.5. Renombrar una tabla

Las sentencias DDL incluyen la sentencia **RENAME**, que es usada para renombrar tablas, vistas, secuencias o sinónimos.

Sintaxis:

```
RENAME old_name TO new_name;
```

Donde:

old_name es el nombre de la tabla, vista, secuencia o sinónimo a renombrar.
new_name es el nuevo nombre de la tabla, vista, secuencia o sinónimo.

Se debe ser dueño del objeto a renombrar.

7.6. Borrar el contenido de una tabla

Oracle dispone de una orden para eliminar definitivamente los datos de una tabla. Se trata de la orden **TRUNCATE TABLE** seguida del nombre de la tabla a borrar. Hace que se elimine el contenido de la tabla, pero no la estructura de la tabla en sí. Incluso borra del archivo de datos el espacio ocupado por la tabla.

Sintaxis

```
TRUNCATE TABLE tabla;
```

Donde:

tabla es el nombre de la tabla

Se debe ser el dueño de la tabla o tener privilegios de sistema DELETE TABLE para truncar una tabla.

La sentencia **DELETE** también puede eliminar todas las filas de la tabla, pero esta sentencia no libera el espacio almacenado.

Eliminar filas con la sentencia TRUNCATE es más rápido que eliminarlas con la sentencia DELETE por las siguientes razones:

- La sentencia TRUNCATE es una sentencia del lenguaje de definición de datos (DDL) y no genera información de transacciones (rollback).
- Truncar una tabla no dispara los TRIGGERS de tipo DELETE de la tabla.
- Si la tabla es padre de una restricción de integridad referencial (constraint), no se puede truncar la tabla. Deshabilite la restricción (constraint) antes de usar la sentencia TRUNCATE.

7.7. Añadir restricciones a una tabla existente

Es posible querer añadir restricciones tras haber creado una tabla. En ese caso se utiliza la siguiente sintaxis:

```
ALTER TABLE tabla
ADD [CONSTRAINT nombre] tipoDeRestricción(columnas);
```

Donde:

tipoRestricción es el texto CHECK, PRIMARY KEY o FOREIGN KEY.

Las restricciones NOT NULL deben añadirse mediante ALTER TABLE .. MODIFY colocando NOT NULL en el campo que se modifica.

Ejemplo:

Añadir la restricción de clave ajena para el supervisor en la tabla de Empleados.

```
ALTER TABLE Empleados
ADD CONSTRAINT emp_supervisor_fk FOREIGN KEY (supervisor_id)
REFERENCES Empleados;
```

7.8. Borrar restricciones

Se puede emplear la cláusula DROP de la sentencia ALTER TABLE para eliminar una restricción. La opción CASCADE hace que se eliminen todas las restricciones dependientes de la elegida.

Sintaxis:

```
ALTER TABLE <tabla>
DROP { PRIMARY KEY | UNIQUE (<columna>) | CONSTRAINT <nombre>
[CASCADE];
```

Donde:

tabla es el nombre de la tabla.
columna es el nombre de la columna afectada por la restricción.
nombre es el nombre de la restricción.

- La opción **PRIMARY KEY** elimina una clave principal (también quitará el índice UNIQUE sobre las campos que formaban la clave.
- La opción **UNIQUE** elimina índices únicos.
- La opción **CONSTRAINT** elimina la restricción indicada.
- La opción **CASCADE** hace que se eliminen en cascada las restricciones de integridad que dependen de la restricción eliminada.

Por ejemplo en:

```
CREATE TABLE curso(
cod_curso CHAR(7) PRIMARY KEY,
fecha_inicio DATE,
fecha_fin DATE,
titulo VARCHAR2(60),
cod_siguientecurso CHAR(7),
CONSTRAINT fecha_ck CHECK(fecha_fin>fecha_inicio),
CONSTRAINT cod_ste_fk FOREIGN KEY(cod_siguientecurso)
REFERENCES curso ON DELETE SET NULL);
```

Tras esa definición de tabla, esta instrucción:

```
ALTER TABLE curso DROP PRIMARY KEY;
```

Produce este error (en Oracle):

ORA-02273: a esta clave única/primaria hacen referencia algunas claves ajenas

Para ello habría que utilizar esta instrucción:

```
ALTER TABLE curso DROP PRIMARY KEY CASCADE;
```

Esa instrucción elimina la restricción de clave secundaria antes de eliminar la principal.

Ejemplo: Eliminar la restricción de clave ajena para el supervisor en la tabla de Empleados.

```
ALTER TABLE Empleados DROP CONSTRAINT emp_supervisor_fk;
```

Eliminar la restricción de clave primaria de la tabla Departamentos y la clave ajena asociada de empleados (Empleados.id-> Departamentos).

```
ALTER TABLE Departamentos DROP PRIMARY KEY CASCADE;
```

7.9. Desactivar restricciones

Se puede desactivar la comprobación de una restricción, aunque siga estando en el esquema, mediante la opción **DISABLE** de la sentencia **ALTER TABLE.**

También se puede incluir DISABLE durante la creación de la tabla (CREATE TABLE) para que desde el principio esté desactivada.

Desactivar una restricción UNIQUE o PRIMARY KEY supone eliminar el índice asociado.

Sintaxis:

```
ALTER TABLE <tabla>
DISABLE CONSTRAINT <nombre> [CASCADE];
```

donde:

tabla es el nombre de la tabla.
nombre es el nombre de la restricción.

Ejemplo:

Desactivar la restricción de clave primaria de la tabla de Empleados.

```
ALTER TABLE Empleados
DISABLE CONSTRAINT emp_pk CASCADE;
```

La opción CASCADE hace que se desactiven también las restricciones dependientes de la que se desactivó.

7.10. Activar restricciones

Se puede activar la comprobación de una restricción mediante la opción **ENABLE** de la sentencia **ALTER TABLE**.

En el momento de la activación, todos los datos de la tabla deben satisfacer la restricción. En el caso de restricciones UNIQUE o PRIMARY KEY, se crea automáticamente un índice asociado.

También se puede incluir ENABLE durante la creación de la tabla (CREATE TABLE).

Sintaxis:

```
ALTER TABLE <tabla>
ENABLE CONSTRAINT <nombre>;
```

donde:

tabla es el nombre de la tabla.
nombre es el nombre de la restricción.

```
Ejemplo:

Volver a activar la restricción de clave primaria de la tabla de
Empleados.

ALTER TABLE Empleados
ENABLE CONSTRAINT emp_pk;
```

Si hubo desactivado en cascada, habrá que activar cada restricción individualmente.

7.11. Cambiar de nombre a las restricciones

Para hacerlo se utiliza este comando (Oracle):

```
ALTER TABLE table
RENAME CONSTRAINT nombreViejo TO nombreNuevo;
```

7. Consultar el diccionario de datos

Las tablas de usuario son tablas creadas por el mismo usuario. Existe otra colección de tablas y vistas en una base de datos Oracle conocidas como *diccionario de datos (data dictionary)*. Esta colección es creada y administrada por el servidor de Oracle y contiene información acerca de la base de datos.

La información almacenada en el diccionario de datos incluye nombres de los usuarios del servidor de Oracle, privilegios otorgados a usuarios, nombres de objetos de la base de datos, reglas de integridad de las tablas e información de auditoria.

Existen 4 categorías de vistas para el diccionario de datos. Cada categoría tiene distintos prefijos que reflejan la intención de su uso.

Prefijo	Descripción
USER_	Estas vistas contienen información acerca de los objetos propiedad del usuario
ALL_	Estas vistas contienen información referente a todas las tablas (tablas y relaciones entre tablas) accesibles al usuario
DBA_	Estas vistas son restringidas, las cuales pueden ser usadas solo por personas con el rol de DBA
V$	Estas vistas son ejecutadas dinámicamente por vistas, por el servidor de base de datos, memoria y bloqueos

Todas las bases de datos disponen de posibilidades para consultar el diccionario de datos. Siguiendo las reglas de Codd, la forma de consultar los metadatos es la misma que en el resto de tablas, es decir, existen tablas (en realidad vistas) que en lugar de contener datos, contienen los metadatos.

8.1. Consultar las tablas de usuario

Oracle utiliza diversas vistas para mostrar las tablas de la base de datos. En concreto **USER_TABLES** que contiene una lista de las tablas del usuario actual (o del esquema actual). Así para sacar la lista de tablas del usuario actual, se haría:

```
SELECT * FROM USER_TABLES;
```

Esta vista tiene numerosas columnas. En concreto, la columna **TABLE_NAME** muestra el nombre de cada tabla. Si deseamos saber las columnas que dispone la vista USER_TABLES dispongo de la orden:

```
DESCRIBE USER_TABLES;
```

Otra vista es **ALL_TABLES** que mostrará una lista de todas las tablas de la base de datos (no solo del usuario actual), aunque oculta las que el usuario no tiene derecho a ver. Finalmente **DBA_TABLES** es una tabla que contiene absolutamente todas las tablas del sistema. Esto es accesible sólo por el usuario administrador (*DBA*).

En el caso de ALL_TABLES y de DBA_TABLES, la columna **OWNER** indica el nombre del propietario de la tabla.

8.2. Obtener la lista de las columnas de las tablas

Otra posibilidad del diccionario de datos es poder consultar las columnas de una tabla.

Oracle posee una vista llamada **USER_TAB_COLUMNS** que permite consultar todas las columnas de las tablas del esquema actual. Las vistas **ALL_TAB_COLUMNS** y **DBA_TAB_COLUMNS** muestran los datos del resto de tablas (la primera sólo de las tablas accesibles por el usuario).

8.3. Mostrar restricciones

Con la sentencia DESCRIBE solo se muestran las restricciones NOT NULL. En Oracle hay dos vistas del diccionario de datos que permiten visualizar la información sobre las restricciones aplicadas en la base de datos.

La vista **USER_CONSTRAINTS** permite identificar las restricciones colocadas por el usuario (ALL_CONSTRAINTS permite mostrar las restricciones de todos los usuarios, pero sólo está permitida a los administradores). En esa vista aparece toda la información que el diccionario de datos posee sobre las restricciones. En ella tenemos las siguientes columnas interesantes:

Columna	Tipo de datos	Descripción
OWNER	VARCHAR2(20)	Indica el nombre del usuario propietario de la tabla
CONSTRAINT_NAME	VARCHAR2(30)	Nombre de la restricción
CONSTRAINT_TYPE	VARCHAR2(1)	Tipo de restricción: • C. De tipo CHECK o NOT NULL • P. PRIMARY KEY • R. FOREIGN KEY • U. UNIQUE
TABLE_NAME	VARCHAR2(30)	Nombre de la tabla en la que se encuentra la restricción

```
Ejemplos:

Obtener la lista de restricciones definidas en el diccionario de
datos:

   SELECT * FROM USER_CONSTRAINTS;

NOT NULL se representa como un caso especial de CHECK.


Obtener la lista de restricciones definidas en el diccionario de
datos para la tabla Empleados:

SELECT * FROM USER_CONSTRAINTS
WHERE TABLE_NAME ='EMPLEADOS';
```

En el diccionario de datos hay otra vista que proporciona información sobre restricciones, se trata de **USER_CONS_COLUMNS.** En dicha vista se muestra información sobre las columnas que participan en una restricción. Así, si hemos definido una clave primaria formada por los campos *uno* y *dos*, en la tabla USER_CONS_COLUMNS aparecerán dos entradas, una para el primer campo del índice y otra para el segundo. Se indicará además el orden de aparición en la restricción. Ejemplo (resultado de la instrucción SELECT * FROM USER_CONS_COLUMNS):

OWNER	CONSTRAINT_NAME	TABLE_NAME	COLUMN_NAME	POSITION
JORGE	EXIS_PK	EXISTENCIAS	TIPO	1
JORGE	EXIS_PK	EXISTENCIAS	MODELO	2
JORGE	EXIS_PK	EXISTENCIAS	N_ALMACEN	3
JORGE	PIEZA_FK	EXISTENCIAS	TIPO	1
JORGE	PIEZA_FK	EXISTENCIAS	MODELO	2
JORGE	PIEZA_PK	PIEZA	TIPO	1
JORGE	PIEZA_PK	PIEZA	MODELO	2

En esta tabla USER_CONS_COLUMNS aparece una restricción de clave primaria sobre la tabla *existencias. E*sta clave está formada por las columnas (*tipo, modelo* y *n_almacen*) y en ese orden. Una segunda restricción llamada *pieza_fk* está compuesta por *tipo* y *modelo* de la tabla existencias. Finalmente la restricción *pieza_pk* está formada por *tipo* y *modelo*, columnas de la tabla pieza.

Para saber de qué tipo son esas restricciones, habría que acudir a la vista USER_CONSTRAINTS.

Otras vistas del diccionario de datos útiles para este tema son ALL_CONSTRAINTS y ALL_CONS_COLUMNS.