DISEÑO FÍSICO DE BASES DE DATOS. LENGUAJE DE DEFINICIÓN DE DATOS

INTRODUCCIÓN

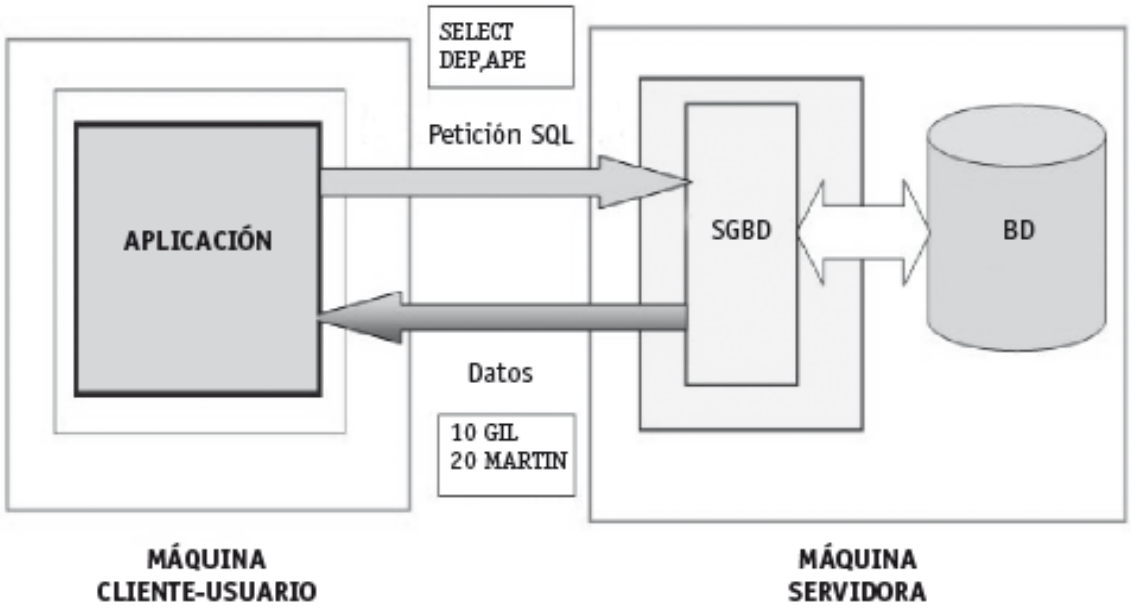
El **lenguaje SQL** (*Structured Query Language*) es una herramienta para organizar, gestionar y recuperar datos almacenados en una base de datos relacional, por tanto, permite la comunicación con el sistema de gestión de la base de datos.
Es tan conocido en bases de datos relacionales que muchos lenguajes de programación incorporan sentencias SQL como parte de su repertorio; tal es el caso de Visual Basic. Entre las principales características del SQL podemos destacar las siguientes:

• Es un lenguaje para todo tipo de usuarios (administradores, desarrolladores y usuarios normales).
• El usuario que emplea SQL especifica qué quiere, no dónde ni cómo.
• Permite hacer cualquier consulta de datos.
• Es posible manejarlo para consultas, actualizaciones, definición de datos y control.

Se puede usar de forma interactiva (el usuario escribe las órdenes desde el teclado de un terminal y al instante obtiene los resultados en la pantalla) y de forma embebida (mezclando las instrucciones SQL con las instrucciones propias del lenguaje, tal es el caso del lenguaje PL/SQL).

La muestra cómo funciona SQL en una arquitectura cliente/servidor. Por un lado, se dispone de una máquina servidora con una base de datos que contiene datos importantes para el negocio. Por otro lado, tenemos una máquina cliente con un usuario que está ejecutando una aplicación que necesita acceder a los datos allí almacenados.

La aplicación realiza una petición al sistema gestor de base de datos, este la procesa y envía los datos a la aplicación que los solicitó.



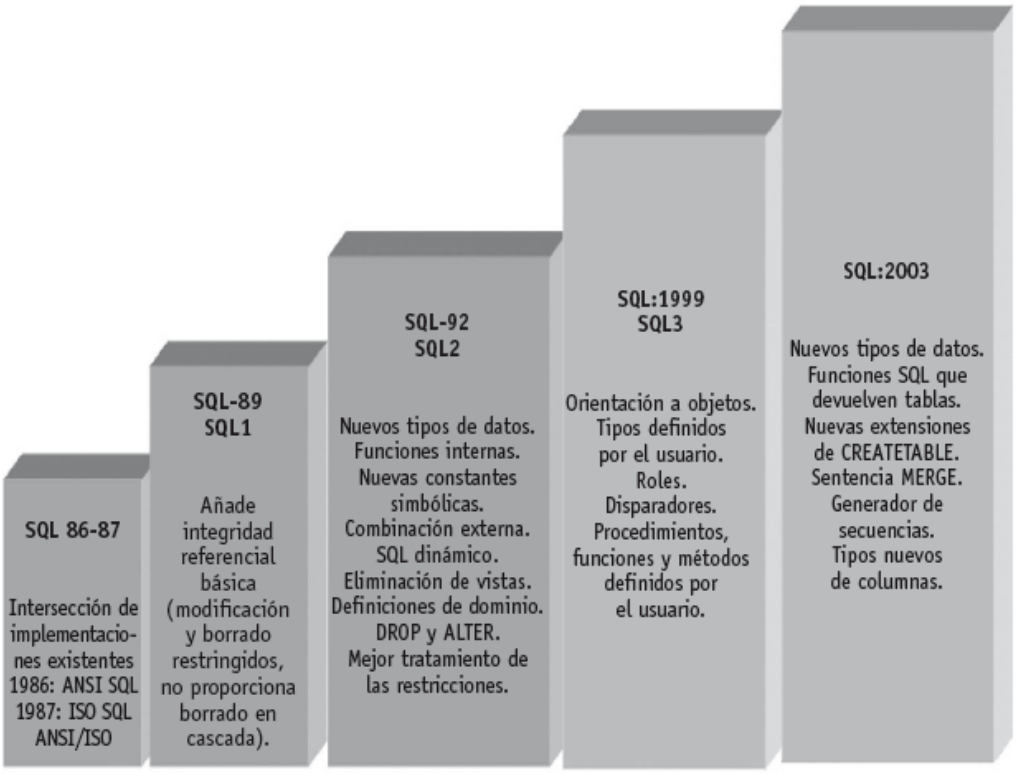
El lenguaje SQL fue desarrollado sobre un prototipo de gestor de bases de datos relacionales denominado SYSTEM R y diseñado por IBM a mediados de los años setenta. Incluía lenguajes de consultas, entre ellos **SEQUEL** (*Structured English Query Language*). Más tarde se renombró como SQL.

En 1979 Oracle Corporation presentó la primera implementación comercial de SQL, que estuvo disponible antes que otros productos de IBM. Por su parte, IBM desarrolló productos herederos del prototipo SYSTEM R, como DB2 y SQL/DS.

El instituto **ANSI** (*American National Standard Institute*) adoptó el lenguaje SQL como estándar para la gestión de bases de datos relacionales en octubre de 1986. En 1987 lo adopta **ISO** (*International Standardization Organization*).

En 1989 el estándar **ANSI/ISO,** revisado y ampliado, se llamó **SQL-89** o estándar **SQL1.** Tres años más tarde se aprueba el estándar **ANSI SQL2** o **SQL-92.** En 1999 se aprueba el estándar **SQL:1999** que introduce mejoras con respecto al anterior. **SQL:2003** es el actual estándar. Revisa todos los apartados de SQL:1999 y añade uno nuevo, el apartado 14: **SQL/XML.**

La Figura a continuación muestra un esquema con la evolución de los estándares y las novedades que va incorporando cada uno con respecto al anterior.



El lenguaje SQL proporciona un gran repertorio de sentencias que se utilizan en variadas tareas, como consultar datos de la base de datos, crear, actualizar y eliminar objetos de la base de datos, crear, actualizar y eliminar datos de los objetos, controlar el acceso a la base de datos y a los objetos. Dependiendo de las tareas, podemos clasificar las sentencias SQL en varios tipos.

	SENTENCIA	DESCRIPCIÓN
DML	***Manipulación de datos***	
	SELECT	Recupera datos de la base de datos.
	INSERT	Añade nuevas filas de datos a la base de datos.
	DELETE	Suprime filas de datos de la base de datos.
	UPDATE	Modifica datos existentes en la base de datos.
DDL	***Definición de datos***	
	CREATE TABLE	Añade una nueva tabla a la base de datos.
	DROP TABLE*	Suprime una tabla de la base de datos.
	ALTER TABLE*	Modifica la estructura de una tabla existente.
	CREATE VIEW*	Añade una nueva vista a la base de datos.
	DROP VIEW*	Suprime una vista de la base de datos.
	CREATE INDEX*	Construye un índice para una columna.
	DROP INDEX*	Suprime el índice para una columna.
	CREATE SYNONYM*	Define un alias para un nombre de tabla.
	DROP SYNONYM*	Suprime un alias para un nombre de tabla.
DCL	***Control de acceso***	
	GRANT	Concede privilegios de acceso a usuarios.
	REVOKE	Suprime privilegios de acceso a usuarios.
	Control de transacciones	
	COMMIT	Finaliza la transacción actual.
	ROLLBACK	Aborta la transacción actual.
	SQL Programático	
	DECLARE	Define un cursor para una consulta.
	OPEN	Abre un cursor para recuperar resultados de consulta.
	FETCH	Recupera una fila de resultados de consulta.
	CLOSE	Cierra un cursor.

* No existían en el estándar SQL1

Tabla 3.1. *Tipos de sentencias SQL.*

Componentes sintácticos de una sentencia

Casi todas las sentencias SQL tienen una forma básica. Véase la Figura Todas comienzan con un verbo, que es una palabra clave que describe lo que hace la sentencia (por ejemplo SELECT, INSERT, UPDATE, CREATE). A continuación, le siguen una o más cláusulas que especifican los datos con los que opera la sentencia. Estas también comienzan con una palabra clave como WHERE o FROM. Algunas son opcionales y otras obligatorias. Muchas contienen nombres de tablas o de columnas, palabras reservadas, constantes o expresiones adicionales.

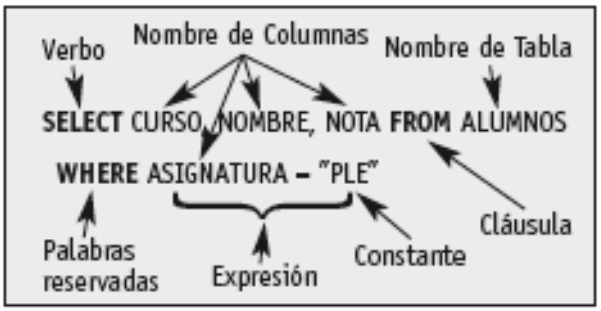


Figura 3.3. *Componentes de una sentencia SQL.*

Cómo procesa el SGBD una sentencia

Para procesar una sentencia SQL el sistema gestor de base de datos recorre una serie de pasos:

1. *Analiza la sentencia.* Comprueba que está sintácticamente bien escrita.

2. *Valida la sentencia.* Comprueba la sentencia semánticamente. Es decir, comprueba si las tablas y las columnas referenciadas en la sentencia existen, si el usuario tiene privilegios para realizar esa operación sobre la tabla, etcétera.

3. *Optimiza la sentencia.* El sistema gestor de base de datos explora la forma de llevar a cabo la ejecución de la sentencia.

4. *Genera un plan de aplicación para la sentencia.* Se genera código ejecutable.

5. *Ejecuta el plan de aplicación*.

El análisis de la sentencia no requiere acceso a la base de datos y, por tanto, suele realizarse rápidamente.
Sin embargo, la optimización es un proceso muy intensivo de CPU y precisa acceder a la base de datos.

Empezaremos creando una tabla. Antes de hacerlo es conveniente planificar ciertos aspectos:

– **El nombre de la tabla.** Debe ser un nombre que identifique su contenido. Por ejemplo, llamamos a una tabla ALUMNOS porque contendrá datos sobre alumnos.

– **El nombre de cada columna de la tabla.** Ha de ser un nombre autodescriptivo, que identifique su contenido. Por ejemplo, DNI, NOMBRE o APELLIDOS.

– **El tipo de dato y el tamaño que tendrá cada columna.**

–**Las columnas obligatorias, los valores por defecto, las restricciones, etcétera.**

La denominación de la tabla puede tener de 1 a 30 caracteres de longitud. Ha de ser única y no puede ser una palabra reservada de Oracle. Su primer carácter debe ser alfabético y el resto pueden ser letras, números y el carácter de subrayado.

Para crear una tabla usamos la orden CREATE TABLE, cuyo formato más simple es:

CREATE TABLE Nombretabla
(
Columna1 Tipo_dato [NOT NULL],
Columna2 Tipo_dato [NOT NULL],
.....
) [TABLESPACE espacio_de_tabla];
Donde:
– Columna1, Columna2 son los nombres de las columnas que contendrá cada fila.
– Tipo_dato indica el tipo de dato (VARCHAR2, NUMBER, etcétera) de cada
columna.
– TABLESPACE espacio_de_tabla señala el TABLESPACE para almacenar
la tabla.
– NOT NULL indica que la columna debe contener alguna información; nunca
puede ser nula.

1 Creamos una tabla llamada ALUMNOS07:

```
CREATE TABLE ALUMNOS07
(
  NUMERO_MATRICULA      NUMBER(6)          NOT NULL,
  NOMBRE                VARCHAR2(15)       NOT NULL,
  FECHA_NACIMIENTO      DATE,
  DIRECCION             VARCHAR2(30),
  LOCALIDAD             VARCHAR2(15)
);
```

Esta sentencia crea una tabla de nombre ALUMNOS07 con cinco columnas llamadas: NUMERO_MATRICULA, NOMBRE, FECHA_NACIMIENTO, DIRECCION y LOCALIDAD. Los tipos de datos para cada columna son: NUMBER, VARCHAR2, DATE, VARCHAR2 y VARCHAR2, respectivamente. La longitud de cada columna es: 6 para el NUMERO_MATRICULA, 15 para el NOMBRE, 30 para la DIRECCION y 15 para la LOCALIDAD. Para el tipo de datos DATE no se define longitud. Oracle la asigna automáticamente.

En las columnas NUMERO_MATRICULA y NOMBRE se ha definido la restricción NOT NULL, indicándose que siempre deben tener algún valor al insertar una nueva fila. Dado que no se ha especificado la cláusula TABLESPACE, la tabla se almacenará en el *tablespace* que tenga asignado el usuario.

La ejecución de la sentencia daría la siguiente salida: `Tabla creada.`

Observaciones:

– Las definiciones individuales de columnas se separan mediante comas.

– No se pone coma después de la última definición de columna.

– Las mayúsculas y minúsculas son indiferentes a la hora de crear una tabla.

Si intentamos crear la tabla ALUMNOS07 y existe otra tabla con este nombre, aparecerá un mensaje de error. Si volvemos a escribir las órdenes anteriores nos dará el siguiente error:

```
CREATE TABLE ALUMNOS07
*
ERROR en línea 1:
ORA-00955: este nombre ya lo está utilizando otro objeto existente
```

Los usuarios pueden consultar las tablas creadas por medio de la vista **USER_TABLES.** Esta vista contiene información acerca de las tablas: nombre de la tabla, nombre del *tablespace*, número de filas, información de almacenamiento, etcétera. Por ejemplo, si queremos visualizar el nombre de las tablas creadas, tendríamos que escribir la orden:

SELECT **TABLE_NAME** FROM **USER_TABLES;**

Existen otras dos vistas que permiten obtener información de los objetos que son propiedad del usuario:

– **USER_OBJECTS:** objetos que son propiedad del usuario.

– **USER_CATALOG:** tablas, vistas, sinónimos y secuencias propiedad del usuario.

Integridad de datos

Cuando almacenamos datos en nuestras tablas, se ajustan a una serie de restricciones predefinidas.
Por ejemplo, que una columna no pueda almacenar valores negativos, que una cadena de caracteres se deba almacenar en mayúsculas o que una columna no pueda ser 0.

La **integridad** hace referencia al hecho de que los datos de la base de datos han de ajustarse a restricciones antes de almacenarse en ella. Así pues, una **restricción de integridad** será una regla que restringe el rango de valores para una o más columnas en la tabla.

Si se produce cualquier fallo mientras un usuario está cambiando los datos en la base de datos, ésta tiene la capacidad de deshacer o cancelar cualquier transacción sospechosa.

Existe otro tipo de integridad, que es la **integridad referencial,** la cual garantiza que los valores de una columna (o columnas) de una tabla (*clave ajena*) dependan de los valores de otra columna (o columnas) de otra tabla (*clave primaria*).

Si en el ejemplo comentado al principio de la unidad sobre las tablas VENTAS y ARTÍCULOS, se define integridad referencial para estas tablas, nunca se dará la situación de insertar una venta con un artículo inexistente. Todas estas restricciones de integridad se explican en el siguiente apartado.

Restricciones en CREATE TABLE

La orden CREATE TABLE permite definir distintos tipos de restricciones sobre una tabla: claves primarias, claves ajenas, obligatoriedad, valores por defecto y verificación de condiciones.

Para definir las restricciones en la orden CREATE TABLE usamos la **cláusula CONSTRAINT.**

Ésta puede restringir una sola columna (*restricción de columna*) o un grupo de columnas de una misma tabla (*restricción de tabla*). Hay dos

modos de especificar restricciones: como parte de la definición de columnas (una restricción de columna) o al final, una vez especificadas todas las columnas (una restricción de tabla).
A continuación, aparece el formato de la orden CREATE TABLE con restricción de columna.

La restricción forma parte de la definición de la columna:

```
CREATE TABLE nombre_tabla
(
Columna1 TIPO_DE_DATO
[CONSTRAINT nombrerestricción]
[NOT NULL] [UNIQUE] [PRIMARY KEY] [DEFAULT valor]
[REFERENCES Nombretabla [(columna [, columna])]
[ON DELETE CASCADE]]
[CHECK (condición)],
Columna2 TIPO_DE_DATO
[CONSTRAINT nombrerestricción]
[NOT NULL] [UNIQUE] [PRIMARY KEY] [DEFAULT valor]
[REFERENCES Nombretabla [(columna [, columna])]
[ON DELETE CASCADE]]
[CHECK (condición)],
...
) [TABLESPACE espacio_de_tabla];
```

Ejemplo:

```
CREATE TABLE EMPLEADO
(
NOMBRE VARCHAR2(25) PRIMARY KEY,
EDAD NUMBER CHECK (EDAD BETWEEN 18 AND 35),
COD_PROVINCIA NUMBER(2) REFERENCES PROVINCIAS ON
DELETE
CASCADE
);
```

Aquí se definen las siguientes restricciones:

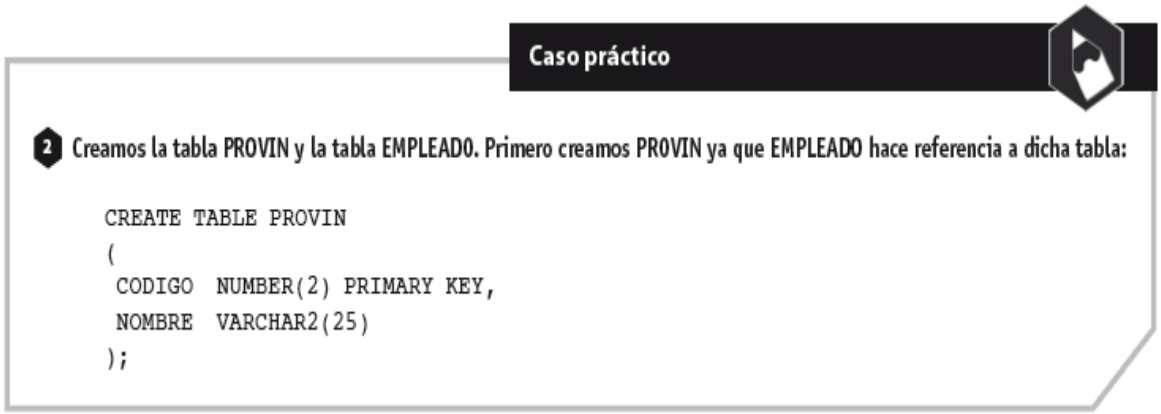
– Clave primaria: NOMBRE.

– Clave ajena: COD_PROVINCIA, que referencia a la tabla PROVINCIAS.

– Verificación de condición CHECK: la edad ha de estar comprendida entre 18 y 35.

Las restricciones de la orden CREATE TABLE que aparecen al final de la definición de las columnas (o de tabla) se diferencian de la anterior en que se puede hacer referencia a varias columnas en una única restricción (por ejemplo, declarando dos columnas como clave primaria o ajena):

CREATE TABLE nombre_tabla
(
Columna1 TIPO_DE_DATO [NOT NULL],
Columna2 TIPO_DE_DATO [NOT NULL],
Columna3 TIPO_DE_DATO [NOT NULL],
...
[**CONSTRAINT** nombrerestricción]
{[UNIQUE] | [PRIMARY KEY] (columna[,columna])},
[**CONSTRAINT** nombrerestricción]
[FOREIGN KEY (columna[,columna])
REFERENCES Nombretabla [(columna[,columna])]
[ON DELETE CASCADE]],
[**CONSTRAINT** nombrerestricción]
[CHECK (condición)],
...
) [TABLESPACE espacio_de_tabla];



Caso práctico

2 Creamos la tabla PROVIN y la tabla EMPLEADO. Primero creamos PROVIN ya que EMPLEADO hace referencia a dicha tabla:

```
CREATE TABLE PROVIN
(
 CODIGO  NUMBER(2) PRIMARY KEY,
 NOMBRE  VARCHAR2(25)
);
```

```
CREATE TABLE EMPLEADO
(
NOMBRE              VARCHAR2(25),
EDAD                NUMBER,
COD_PROVINCIA       NUMBER(2),
CONSTRAINT          PK_EMPLEADO PRIMARY KEY (NOMBRE),
CONSTRAINT          CK_EDAD     CHECK(EDAD BETWEEN  18 AND  35),
CONSTRAINT          FK_EMPLEADO FOREIGN KEY  (COD_PROVINCIA)
                    REFERENCES PROVIN ON DELETE  CASCADE
);
```

El nombre de las restricciones es opcional. También es válida esta sentencia CREATE TABLE:

```
CREATE TABLE EMPLEADO
(
NOMBRE              VARCHAR2(25),
EDAD                NUMBER(2),
COD_PROVINCIA       NUMBER(2),
PRIMARY KEY (NOMBRE),
CHECK (EDAD BETWEEN 18 AND 35),
FOREIGN KEY (COD_PROVINCIA) REFERENCES PROVIN
                            ON DELETE CASCADE
);
```

Clave primaria. La restricción PRIMARY KEY

Una **clave primaria dentro de una tabla** es una columna o un conjunto de columnas que identifican unívocamente a cada fila. Debe ser única, no nula y obligatoria.

Como máximo podemos definir una clave primaria por tabla. Esta clave se puede referenciar por una columna o columnas de otra tabla. Llamamos clave ajena a esta columna o columnas.

Cuando se crea una clave primaria, automáticamente se crea un índice que facilita el acceso a la tabla. Para definir una clave primaria en una tabla usamos la restricción **PRIMARY KEY.**

Éstos son los formatos de la orden CREATE TABLE para definir claves primarias:

– El **formato de restricción de columna** es el siguiente:
CREATE TABLE nombre_tabla
(
Columna1 TIPO_DE_DATO [CONSTRAINT nombrerestricción]
PRIMARY KEY,
Columna2 TIPO_DE_DATO,
...
) [TABLESPACE espacio_de_tabla];

– Y el **formato de restricción de tabla** es éste:

CREATE TABLE nombre_tabla
(
Columna1 TIPO_DE_DATO,
Columna2 TIPO_DE_DATO,
...
[CONSTRAINT nombrerestricción] **PRIMARY KEY** (columna [, columna]),
...
) [TABLESPACE espacio_de_tabla];

Caso práctico

3 **Creamos la tabla BLOQUESPISOS. Las columnas son las siguientes:**

Nombre columna	Representa	Tipo
CALLE	Calle donde está el bloque	VARCHAR2(30)
NUMERO	Número donde está el bloque	NUMBER(3)
PISO	Número de planta	NUMBER(2)
PUERTA	Puerta	CHAR(1)
CODIGO_POSTAL	Código postal	NUMBER(5)
METROS	Metros de la vivienda	NUMBER(5)
COMENTARIOS	Otros datos de la vivienda	VARCHAR2(60)
COD_ZONA	Código de zona donde está el bloque	NUMBER(2)
DNI	DNI del propietario	VARCHAR2(10)

La clave primaria estará formada por las columnas CALLE, NUMERO, PISO y PUERTA que, por tanto, no pueden contener valores nulos. Se puede crear la tabla de la siguiente forma:

```
CREATE TABLE BLOQUESPISOS
(
 CALLE              VARCHAR2(30)       NOT NULL,
 NUMERO             NUMBER(3)          NOT NULL,
 PISO               NUMBER(2)          NOT NULL,
 PUERTA             CHAR(1)            NOT NULL,
 CODIGO_POSTAL      NUMBER(5),
 METROS             NUMBER(5),
 COMENTARIOS        VARCHAR2(60),
 COD_ZONA           NUMBER(2),
 DNI                VARCHAR2(10),
 CONSTRAINT PK_VIV PRIMARY KEY (CALLE, NUMERO, PISO, PUERTA)
);
```

La última sentencia se podría haber puesto de la siguiente manera: **PRIMARY KEY (CALLE, NUMERO, PISO, PUERTA)**, sin dar nombre a la restricción.

La clave de esta tabla es la combinación de CALLE, NUMERO, PISO y PUERTA, que están especificadas como NOT NULL. Esto permite evitar la introducción de datos en la tabla sin dar valores a determinadas columnas. Si no ponemos NOT NULL en alguno de los atributos de la clave, Oracle automáticamente coloca NOT NULL en ese atributo. Al visualizar la descripción de la tabla comprobamos que estas cuatro columnas tienen la restricción NOT NULL.

Creamos la tabla ZONAS. Las columnas para esta tabla son:

Nombre columna	Representa	Tipo
COD_ZONA	Código de la zona	NUMBER(2)
NOMBREZONA	Nombre de zona	VARCHAR2(20)
MASDATOS	Otros datos de la zona	VARCHAR2(50)

La clave primaria es el código de zona (COD_ZONA) y la definimos formando parte de la columna (restricción de columna):

```
CREATE TABLE ZONAS
(
      COD_ZONA      NUMBER(2)          PRIMARY KEY,
      NOMBREZONA    VARCHAR2(15)       NOT NULL,
      MASDATOS      VARCHAR2(60)
);
```

Si en una tabla forman parte de la clave primaria varias columnas, ésta no se puede definir como restricción de columna. Cuando en la orden CREATE TABLE aparece la cláusula PRIMARY KEY sólo se debe especificar una vez.

Claves ajenas. La restricción FOREIGN KEY

Una clave ajena está formada por una o varias columnas que están asociadas a una clave primaria de otra o de la misma tabla. Se pueden definir tantas claves ajenas como sea preciso, y pueden estar o no en la misma tabla que la clave primaria.

El valor de la columna o columnas que son claves ajenas debe ser NULL o igual a un valor de la clave
referenciada (regla de integridad referencial).

El formato de CREATE TABLE para definir claves ajenas puede ser cualquiera de los siguientes:

– **Formato de restricción de columna.** La clave ajena se define en la descripción de la columna usando la cláusula REFERENCES:

CREATE TABLE nombre_tabla
(
Columna1 TIPO_DE_DATO
[CONSTRAINT nombrerestricción]
REFERENCES Nombretabla [(columna)] [ON DELETE CASCADE],
...

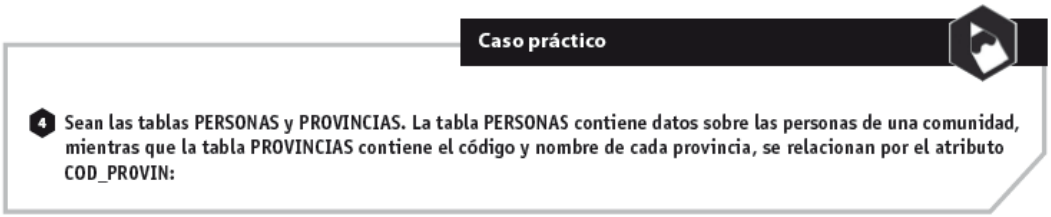
Columna2 TIPO_DE_DATO, ...
) [TABLESPACE espacio_de_tabla];

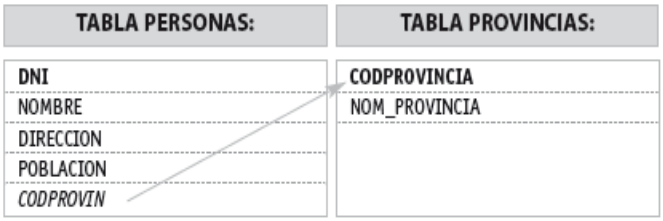
– **Formato de restricción de tabla.** La clave ajena se define al final de todas las columnas empleando las cláusulas FOREIGN KEY y REFERENCES:

CREATE TABLE nombre_tabla
(
Columna1 TIPO_DE_DATO,
Columna2 TIPO_DE_DATO,
...
[CONSTRAINT nombrerestricción] **FOREIGN KEY** (columna
[, columna]) **REFERENCES** Nombretabla [(columna[,
columna])] [ON DELETE CASCADE],
...
) [TABLESPACE espacio_de_tabla];

En la cláusula **REFERENCES** indicamos la tabla a la cual remite la clave ajena. La derecha de FOREIGN KEY y, entre paréntesis, indicamos la columna o columnas que forman parte de la clave ajena.

La cláusula **ON DELETE CASCADE** o **borrado en cascada** se define cuando al borrar las filas asociadas con claves primarias deseamos que se eliminen automáticamente las filas con claves ajenas que referencien a dichas claves.





Donde:

- DNI es la clave primaria de la tabla PERSONAS.
- CODPROVINCIA de la tabla PROVINCIAS es clave primaria de esta tabla.
- CODPROVIN de la tabla PERSONAS es clave ajena, porque se relaciona con la clave primaria de la tabla PROVINCIAS. Los valores que se almacenen en esta columna deben coincidir con la clave primaria de la tabla PROVINCIAS. Se puede afirmar que PROVINCIAS es la tabla maestra y PERSONAS es la tabla detalle.

Hemos de crear, en primer lugar, la tabla PROVINCIAS y, después, la tabla PERSONAS, ya que PERSONAS referencia a PROVINCIAS. Si creamos primero la tabla PERSONAS y la tabla PROVINCIAS no está creada, Oracle emitirá un mensaje de error.

```
CREATE TABLE PROVINCIAS
(
    CODPROVINCIA   NUMBER(2)      PRIMARY KEY,
    NOM_PROVINCIA  VARCHAR2(15)
);
CREATE TABLE PERSONAS
(
    DNI            NUMBER(8)      PRIMARY KEY,
    NOMBRE         VARCHAR2(15),
    DIRECCION      VARCHAR2(25),
    POBLACION      VARCHAR2(20),
    CODPROVIN      NUMBER(2) NOT NULL REFERENCES PROVINCIAS
);
```

La clave ajena se ha definido usando la cláusula REFERENCES, aunque también se puede definir usando la cláusula FOREIGN KEY de la siguiente manera:

```
CREATE TABLE PERSONAS
(
    DNI            NUMBER(8)      PRIMARY KEY,
    NOMBRE         VARCHAR2(15),
    DIRECCION      VARCHAR2(25),
    POBLACION      VARCHAR2(20),
    CODPROVIN      NUMBER(2)      NOT NULL,
    FOREIG KEY   (CODPROVIN) REFERENCES PROVINCIAS
);
```

Caso práctico

5 **Partimos de una situación en que las dos tablas tienen datos. Vamos a borrar todas las filas de la tabla PROVINCIAS:**

```
DELETE PROVINCIAS;
*
ERROR en línea 1:
ORA-02292: restricción de integridad (SCOTT.SYS_C005389) violada - registro secun-
dario encontrado
```

Se produce un error: no podemos borrar filas en la tabla maestra (PROVINCIAS) si hay filas en la tabla detalle (PERSONAS) que las estén referenciando. Es decir, una fila no se puede borrar si es referenciada por alguna clave ajena.

Si se añade la cláusula ON DELETE CASCADE en la opción REFERENCES de la tabla detalle (PERSONAS) se podrán eliminar las filas de la tabla maestra (PROVINCIAS) y las filas correspondientes en la tabla detalle (PERSONAS) con esa provincia serán eliminadas. Borramos la tabla PERSONAS de la siguiente manera: DROP TABLE PERSONAS; y la volvemos a crear así:

```
CREATE TABLE PERSONAS
(
    DNI             NUMBER(8)          PRIMARY KEY,
    NOMBRE          VARCHAR2(15),
    DIRECCION       VARCHAR2(25),
    POBLACION       VARCHAR2(20),
    CODPROVIN       NUMBER(2)          NOT NULL,
    FOREIGN KEY     (CODPROVIN) REFERENCES PROVINCIAS ON DELETE CASCADE
);
```

Como veremos más adelante, DROP TABLE se utiliza para suprimir una tabla de la base de datos.

Una vez creada la tabla insertamos filas y borramos una fila de la tabla maestra (PROVINCIAS). Automáticamente se borrarán las filas de la tabla detalle que se correspondan con las filas de la tabla maestra. Esta acción mantiene automáticamente la integridad referencial.

A la hora de borrar tablas relacionadas, primero se ha de borrar la tabla detalle y después la tabla maestra. Si se borra la tabla maestra antes que la tabla detalle se producirá un error:

```
DROP TABLE PROVINCIAS;
*
ERROR en línea 1:
ORA-02449: claves únicas/primarias en la tabla referidas por claves ajenas
```

De esta manera, se indica que hay claves ajenas que hacen referencia a esta tabla.

Hasta ahora, al definir las restricciones de clave primaria y clave ajena, no les hemos dado nombre. Por defecto, Oracle asigna un nombre a la restricción SYS_C00n, donde '*n*' es un número asignado automáticamente por Oracle.

En el siguiente ejemplo se intenta insertar una fila en la tabla PROVINCIAS cuyo código de provincia (clave primaria) ya existe:

```
INSERT INTO PROVINCIAS VALUES(6,'CÁCERES');
*
ERROR en línea 1:
ORA-00001: restricción única (SCOTT.SYS_C005386) violada
```

Oracle da un mensaje de error: la restricción que ha sido violada es la SYS_C005386, que hace referencia a la violación de la clave primaria.

El propietario de la tabla es el usuario llamado SCOTT.
En el siguiente ejemplo se pretende insertar una fila en la tabla PERSONAS cuyo código de provincia (clave ajena) no existe en la tabla PROVINCIAS:

```
INSERT INTO PERSONAS VALUES(1122, 'Pedro', 'La Peña 16',
'Berrocalejo', 25);
*
ERROR en línea 1:
ORA-02291: restricción de integridad (SCOTT.SYS_C005392)
violada - clave principal no encontrada
```

La restricción que se ha violado ahora es la SYS_C005392, que hace referencia a la inexistencia del valor de la clave primaria en la tabla PROVINCIAS al que remite la clave ajena.

El nombre de una restricción es un nombre único que define el propietario del objeto o, por defecto, el sistema. Se asigna en el momento de definir la restricción. Por defecto, la denominación es SYS_C00n. La cláusula que da nombre a la restricción es la siguiente:

CONSTRAINT nombrerestricción.

Caso práctico

6 **Borramos las tablas PERSONAS y PROVINCIAS: `DROP TABLE PERSONAS; DROP TABLE PROVINCIAS;` y creamos las tablas de nuevo dando nombre a las restricciones de clave primaria y clave ajena:**

```
CREATE TABLE PROVINCIAS
(
    CODPROVINCIA    NUMBER(2)      CONSTRAINT PK_PROV PRIMARY KEY,
    NOM_PROVINCIA   VARCHAR2(15)
);
```

PK_PROV es el nombre de la restricción de clave primaria.

```
CREATE TABLE PERSONAS
(
    DNI             NUMBER(8)      CONSTRAINT PK_PER PRIMARY KEY,
    NOMBRE          VARCHAR2(15),
    DIRECCION       VARCHAR2(25),
    POBLACION       VARCHAR2(20),
    CODPROVIN       NUMBER(2) NOT NULL,
    CONSTRAINT FK_PER FOREIGN KEY (CODPROVIN) REFERENCES PROVINCIAS
                    ON DELETE  CASCADE
);
```

PK_PER es el nombre de la restricción de clave primaria. FK_PER es el nombre de la restricción de clave ajena.

A continuación se insertarán algunas filas en la tabla PROVINCIAS para hacer que se disparen las restricciones:

```
INSERT INTO PROVINCIAS VALUES(28, 'MADRID');
INSERT INTO PROVINCIAS VALUES(28,'SEVILLA');
*
ERROR en línea 1:
ORA-00001: restricción única (SCOTT.PK_PROV) violada
```

Resulta más fácil identificar una violación de una restricción si se le da un nombre al definirla.

Al insertar un código de provincia existente se produce un error en la restricción de clave primaria llamada PK_PROV. Ahora insertamos una fila en la tabla PERSONAS cuyo código de provincia no exista en la tabla PROVINCIAS:

```
INSERT INTO PERSONAS VALUES(1133, 'Luis', 'La Peña 12',  'Berrocalejo', 22);

ERROR en línea 1:
ORA-02291: restricción de integridad (SCOTT.FK_PER) violada - clave principal no
encontrada
```

Se produce un error en la restricción de clave ajena llamada FK_PER.

Obligatoriedad. La restricción NOT NULL

Esta restricción asociada a una columna significa que no puede tener valores nulos, es decir, que ha de tener obligatoriamente un valor. En caso contrario, causa una excepción.

En ejemplos anteriores nos hemos ocupado de cómo se define una columna con la restricción NOT NULL. Éste es su formato:

CREATE TABLE nombre_tabla
(
```
Columna1 TIPO_DE_DATO [CONSTRAINT nombrerestricción]
NOT NULL,
Columna2 TIPO_DE_DATO
...
) [TABLESPACE espacio_de_tabla];
```

Caso práctico

7 **Creamos una tabla definiendo las columnas NIF y NOMBRE como no nulas y damos nombre a la restricción no nula de la columna NOMBRE:**

```
CREATE TABLE EJEMPLO
(
  NIF       VARCHAR2(10)   NOT NULL,
  NOMBRE    VARCHAR(30)    CONSTRAINT NOMNONULO NOT NULL,
  EDAD      NUMBER(2)
);

INSERT INTO EJEMPLO (NIF) VALUES ('45222111-A');
*
ERROR en línea 1:
ORA-01400: no se puede realizar una inserción NULL en
("SCOTT"."EJEMPLO"."NOMBRE")
```

Cuando se viola una columna NOT NULL se produce una excepción y, aunque se dé nombre a una restricción NOT NULL, no aparecerán los mensajes de restricción vistos antes, en los que aparecía el nombre de la restricción violada.

Valores por defecto. La especificación DEFAULT

En el momento de crear una tabla podemos asignar valores por defecto a las columnas.

Si especificamos la cláusula DEFAULT a una columna, le proporcionamos un valor por omisión cuando el valor de la columna no se especifica en la cláusula INSERT.

En la especificación DEFAULT es posible incluir varias expresiones: constantes, funciones SQL y variables UID y SYSDATE. No se puede hacer referencias a columnas o a funciones PL/SQL.

Caso práctico

8 Se crea la tabla EJEMPLO1 y se asigna a la columna FECHA la fecha del sistema:

```
CREATE TABLE EJEMPLO1
(
  DNI          VARCHAR2(10) NOT NULL,
  NOMBRE       VARCHAR(30),
  FECHA        DATE DEFAULT SYSDATE
);
```

Se inserta una fila en la tabla dando valores a todas las columnas, salvo a la columna FECHA: `INSERT INTO EJEMPLO1(DNI, NOMBRE) VALUES('1234','PEPA');` Al visualizar el contenido de la tabla, en la columna FECHA se almacenará la fecha del sistema, ya que no se dio valor.

Verificación de condiciones. La restricción CHECK

Muchas columnas de tablas requieren valores limitados dentro de un rango o el cumplimiento de ciertas condiciones. Con una restricción de verificación de condiciones se puede expresar una condición que ha de cumplirse para todas y cada una de las filas de la tabla.

La **restricción CHECK** actúa como una cláusula WHERE. Puede hacer referencia a una o a más columnas, pero no a valores de otras filas. En una cláusula CHECK no cabe incluir subconsultas ni las pseudocolumnas SYSDATE, UID y USER.
Éstos son los formatos con la orden CREATE TABLE con la restricción CHECK:

– Formato de restricción de columna:

CREATE TABLE nombre_tabla
(
Columna1 TIPO_DE_DATO [**CONSTRAINT** nombrerestricción]
CHECK (condición),
Columna2 TIPO_DE_DATO
...
) [TABLESPACE espacio_de_tabla];

Formato de restricción de tabla:
CREATE TABLE nombre_tabla
(
Columna1 TIPO_DE_DATO,
Columna2 TIPO_DE_DATO,
...

[**CONSTRAINT** nombrerestricción] **CHECK** (condición),
...
) [TABLESPACE espacio_de_tabla];

Caso práctico

9 Se crea la tabla EJEMPLO3. Las columnas son: DNI VARCHAR2(10), NOMBRE VARCHAR2(30), EDAD NUMBER(2), CURSO NUMBER; y las restricciones:

- El DNI no puede ser nulo.
- La clave primaria es el DNI.
- El NOMBRE no puede ser nulo.
- La EDAD ha de estar comprendida entre 5 y 20 años.
- El NOMBRE ha de estar en mayúsculas.
- El CURSO sólo puede almacenar 1, 2 ó 3.

Es posible crear la tabla de varias maneras: se puede dar nombre o no a las restricciones, definir las restricciones en la descripción de la columna o al final, combinando ambas, etcétera:

```
CREATE TABLE EJEMPLO3
(
  DNI           VARCHAR2(10)     NOT NULL,
  NOMBRE        VARCHAR2(30)     NOT NULL,
  EDAD          NUMBER(2),
  CURSO         NUMBER,
  CONSTRAINT    CLAVE_P          PRIMARY KEY(DNI),
  CONSTRAINT    COMP_EDAD        CHECK (EDAD BETWEEN 5 AND 20),
  CONSTRAINT    NOMBRE_MAYUS     CHECK (NOMBRE = UPPER(NOMBRE)),
  CONSTRAINT    COMP_CURSO       CHECK (CURSO IN(1, 2, 3))
);
```

Sin dar nombre a las restricciones y definiéndolas en la descripción de las columnas, tenemos:

```
CREATE TABLE EJEMPLO3
(
  DNI           VARCHAR2(10)     NOT NULL PRIMARY KEY,
  NOMBRE        VARCHAR2(30)     NOT NULL CHECK (NOMBRE = UPPER(NOMBRE)),
  EDAD          NUMBER(2)        CHECK (EDAD BETWEEN 5 AND 20),
  CURSO         NUMBER           CHECK (CURSO IN(1, 2, 3))
);
```

La restricción NOT NULL es similar a: **CHECK**(nombre_columna **IS NOT NULL**).

La restricción UNIQUE

Evita valores repetidos en la misma columna. Puede contener una o varias columnas. Es similar a la restricción PRIMARY KEY, salvo que son posibles varias columnas UNIQUE definidas en una tabla. Admite valores NULL. Al igual que en PRIMARY KEY, cuando se define una restricción UNIQUE se crea un índice automáticamente. Veamos su formato:

– Formato de restricción de columna:

CREATE TABLE nombre_tabla
(
Columna1 TIPO_DE_DATO [**CONSTRAINT** nombrerestricción]
UNIQUE,
Columna2 TIPO_DE_DATO
...
) [TABLESPACE espacio_de_tabla];
– Formato de restricción de tabla:
CREATE TABLE nombre_tabla
(
Columna1 TIPO_DE_DATO,
Columna2 TIPO_DE_DATO,
...
[**CONSTRAINT** nombrerestricción] **UNIQUE** (columna [,
columna]),
...
) [TABLESPACE espacio_de_tabla];

Caso práctico

10 Se crea la tabla EJEMPLO1_U definiendo una columna con UNIQUE y se intentan insertar dos filas, una de ellas violando la restricción.

```
CREATE TABLE EJEMPLO1_U
(
  DNI     VARCHAR2(10)      PRIMARY KEY,
  NOM     VARCHAR2(30)      UNIQUE,
  EDAD    NUMBER(2)
);
```

Se inserta la primera fila:

```
INSERT INTO EJEMPLO1_U VALUES ('11111', 'PEPA', 20);
```

Se inserta la segunda fila:

```
INSERT INTO EJEMPLO1_U  VALUES ('11112', 'PEPA', 21);
*
ERROR en línea 1:
ORA-00001: restricción única (SCOTT.SYS_C005401) violada
```

Vistas del diccionario de datos para las restricciones

Existe una serie de vistas creadas por Oracle que contienen información referente a las restricciones definidas en las tablas. Contienen información general las siguientes:

– USER_CONSTRAINTS: definiciones de restricciones de tablas propiedad del usuario.

– ALL_CONSTRAINTS: definiciones de restricciones sobre tablas a las que puede acceder el usuario.

– DBA_CONSTRAINTS: todas las definiciones de restricciones sobre todas las tablas.

El tipo de restricción, columna CONSTRAINT_TYPE de estas vistas, puede ser:

C Restricciones de tipo CHECK	R Restricción FOREIGN KEY (References)
P Restricción PRIMARY KEY	U Restricción UNIQUE

Para información sobre restricciones en las columnas tenemos:

– USER_CONS_COLUMNS: información sobre las restricciones de columnas en tablas del usuario.
– ALL_CONS_COLUMNS: información sobre las restricciones de columnas en tablas a las que puede acceder el usuario.
– DBA_CONS_COLUMNS: información sobre todas las restricciones de columnas.

Caso práctico

11 **Observemos las restricciones de la tabla EJEMPLO: el nombre de restricción, el nombre de la tabla y el tipo de restricción:** `SELECT CONSTRAINT_NAME, TABLE_NAME, CONSTRAINT_TYPE FROM USER_CONSTRAINTS WHERE TABLE_NAME = 'EJEMPLO';`

Para ver las restricciones definidas en las columnas de la tabla 'EJEMPLO' se teclea la orden siguiente: `SELECT CONSTRAINT_NAME, TABLE_NAME, COLUMN_NAME FROM USER_CONS_COLUMNS WHERE TABLE_NAME = 'EJEMPLO';`

Creación de una tabla con datos recuperados en una consulta

La sentencia CREATE TABLE permite crear una tabla a partir de la consulta de otra tabla ya existente.
La nueva tabla contendrá los datos obtenidos en la consulta. Se lleva a cabo esta acción con la cláusula AS colocada al final de la orden CREATE TABLE. El formato es el que sigue:

CREATE TABLE Nombretabla
(
Columna [, Columna]
)
[TABLESPACE espacio_de_tabla]
AS consulta;

No es necesario especificar tipos ni tamaño de las columnas, ya que vienen determinados por los tipos y los tamaños de las recuperadas en la consulta.
La consulta puede contener una subconsulta, una combinación de tablas o cualquier sentencia SELECT válida.

Las restricciones CON NOMBRE no se crean en una tabla desde la otra, sólo se crean aquellas restricciones que carecen de nombre.

Caso práctico

12 **Para crear la tabla EJEMPLO_AS a partir de los datos de la tabla EJEMPLO se procede así:**

```
CREATE TABLE EJEMPLO_AS
AS SELECT * FROM EJEMPLO;
```

La tabla se crea con los mismos nombres de columnas e idéntico contenido de filas que la tabla EJEMPLO.

En el siguiente ejemplo, asignamos un nombre a las columnas de la tabla EJEMPLO_AS2 (COL1, COL2, COL3 y COL4):

```
CREATE TABLE EJEMPLO_AS2 (COL1, COL2, COL3, COL4)
AS SELECT * FROM EJEMPLO;
```

El contenido de la tabla es el mismo que el que tiene la tabla EJEMPLO.

Se crea la tabla EMPLEYDEPART a partir de las tablas EMPLE y DEPART. Esta tabla contendrá el apellido y el nombre del departamento de cada empleado:

```
CREATE TABLE EMPLEYDEPART
AS SELECT APELLIDO, DNOMBRE
FROM EMPLE, DEPART WHERE EMPLE.DEPT_NO = DEPART.DEPT_NO;
```

A continuación, se listan las restricciones para las tablas EJEMPLO y EJEMPLO_AS: `SELECT CONSTRAINT_NAME, TABLE_NAME FROM USER_CONSTRAINTS WHERE TABLE_NAME IN ('EJEMPLO', 'EJEMPLO_AS');` Observamos que en la tabla EJEMPLO_AS sólo aparecen dos restricciones sin nombre, correspondientes a la restricción NOT NULL y definidas para las dos primeras columnas de la tabla, que son el DNI y el NOMBRE.

SUPRESION DE TABLAS

La orden SQL DROP TABLE suprime una tabla de la base de datos. Cada usuario puede borrar sus propias tablas; sólo el administrador de la base de datos o algún usuario con el privilegio DROP ANY TABLE puede borrar las tablas de otro usuario.

Al suprimir una tabla también se suprimen los índices y los privilegios asociados a ella. Las vistas y los
sinónimos creados a partir de esta tabla dejan de funcionar, pero siguen existiendo en la base de datos, por lo que habría que eliminarlos. El formato de la orden DROP TABLE es:

DROP TABLE [usuario].nombretabla [CASCADE CONSTRAINTS];
CASCADE CONSTRAINTS elimina las restricciones de integridad referencial que remitan a la clave primaria de la tabla borrada.

Caso práctico

13 **Recordemos ahora la tabla PROVINCIAS, que tiene definida clave primaria en la columna CODPROVINCIA, y la tabla PERSONAS, que tiene definida una clave ajena (CODPROVIN) referenciando a la tabla PROVINCIAS.**

Si intentamos borrar la tabla PROVINCIAS, Oracle nos dará un mensaje de error:

```
DROP TABLE PROVINCIAS;
           *
ERROR en línea 1:
ORA-02449: claves únicas/primarias en la tabla referidas por claves ajenas
```

El error se debe a que existe una restricción de clave ajena en la tabla PERSONAS que referencia a la clave primaria de la tabla PROVINCIAS. Para borrar esta tabla hay que usar la opción CASCADE CONSTRAINTS, que suprime todas las restricciones de integridad referencial que se refieran a claves de la tabla borrada:

```
DROP TABLE PROVINCIAS CASCADE CONSTRAINTS;
```

Orden TRUNCATE

Permite suprimir todas las filas de una tabla y liberar el espacio ocupado para otros usos sin que desaparezca la definición de la tabla de la base de datos.

Es una orden del lenguaje de definición de datos o DDL y no genera información de retroceso (ROLLBACK);
es decir, una sentencia TRUNCATE no se puede anular, como tampoco activa disparadores DELETE. Por eso, la eliminación de filas con la orden TRUNCATE es más rápida que con DELETE.

Su formato es:

TRUNCATE TABLE [usuario.]nombretabla [{DROP|REUSE} STORAGE];
La siguiente sentencia borra todas las filas de la tabla EJEMPLO:
TRUNCATE TABLE EJEMPLO;

De forma opcional, TRUNCATE permite liberar el espacio utilizado por las filas suprimidas.
Con la opción DROP STORAGE se libera todo el espacio, excepto el especificado mediante el parámetro MINEXTENTS de la tabla; se trata de la opción por defecto. REUSE STORAGE mantendrá reservado el espacio para nuevas filas de la tabla.
No se puede truncar una tabla cuya clave primaria sea referenciada por la clave ajena de otra tabla. Antes de truncar la tabla hay que desactivar la restricción.

Ejemplo:

TRUNCATE TABLE PROVINCIAS;
ERROR en línea 1:
ORA-02266: claves únicas/primarias en la tabla referidas
por claves ajenas activadas

MODIFICACIÓN DE TABLAS

Se puede modificar una tabla mediante la orden **ALTER TABLE.** La modificación de tablas nos permitirá: añadir, modificar o eliminar columnas de una tabla existente, añadir o eliminar restricciones y activar o desactivar restricciones.

ALTER TABLE nombretabla
{[**ADD** (columna)
[**MODIFY** (columna [,columna] ...)]
[**DROP COLUMN** (columna [,columna] ...)]
[**ADD CONSTRAINT** restricción]
[**DROP CONSTRAINT** restricción]
[**DISABLE CONSTRAINT** restricción]
[**ENABLE CONSTRAINT** restricción]};

Añadir, modificar o eliminar columnas a una tabla

Veamos a continuación cómo se usa la orden ALTER TABLE para añadir, modificar o eliminar columnas de una tabla.

• **Add:** Se utiliza ADD para añadir columnas a una tabla. A la hora de añadir una columna a una tabla hay que tener en cuenta varios factores:

– Si la columna no está definida como NOT NULL, se le puede añadir en cualquier momento.

– Si la columna está definida como NOT NULL, cabe la posibilidad de seguir los siguientes pasos: en primer lugar se añade la columna sin especificar NOT NULL; después se da valor a la columna para cada una de las filas; finalmente, se modifica la columna a NOT NULL.

• **Modify:** Modifica una o más columnas existentes en la tabla. Al modificar una columna de una tabla se han de tener en cuenta estos aspectos:

– Se puede aumentar la longitud de una columna en cualquier momento.

– Al disminuir la longitud de una columna que tiene datos no se puede dar menor tamaño que el máximo valor almacenado.

– Es posible aumentar o disminuir el número de posiciones decimales en una columna de tipo NUMBER.

– Si la columna es NULL en todas las filas de la tabla, se puede disminuir la longitud y modificar el tipo de dato.

– La opción MODIFY ... NOT NULL sólo será posible cuando la tabla no contenga ninguna fila con valor nulo en la columna que se modifica.

• **Drop column:** Se utiliza para borrar una columna de una tabla. Hay que tener en cuenta que no se pueden borrar todas las columnas de una tabla y tampoco se pueden eliminar claves primarias referenciadas por claves ajenas.

Adición y borrado de restricciones

Podemos añadir y eliminar las siguientes restricciones de una tabla: CHECK, PRIMARY KEY, NOT NULL, FOREIGN KEY y UNIQUE.

Para añadir restricciones usamos la orden: ALTER TABLE nombretabla **ADD CONSTRAINT** nombrerestricción ...

Para eliminar restricciones usamos la orden: ALTER TABLE nombretabla **DROP CONSTRAINT** nombrerestriccion ...

Se pueden eliminar las restricciones con nombre y las asignadas por el sistema (SYS_C00n).

Casos prácticos

14 **Supongamos que la tabla EJEMPLO3 tiene datos y queremos añadir dos columnas: SEXO con la restricción NOT NULL e IMPORTE. Ocurrirá un error ya que la tabla no está vacía:**

```
ALTER TABLE EJEMPLO3 ADD(SEXO CHAR(1) NOT NULL, IMPORTE NUMBER(4));
ERROR en línea 1:
ORA-01758: la tabla debe estar vacía para agregar la columna (NOT NULL)
obligatoria
```

La opción ADD ... NOT NULL sólo será posible si la tabla está vacía. La solución a este problema consiste en modificar la tabla añadiendo la columna sin restricción:

```
ALTER TABLE EJEMPLO3 ADD (SEXO CHAR(1), IMPORTE NUMBER(4));
```

A continuación, modifica la columna dándole un valor, sea verdadero o no:

```
UPDATE EJEMPLO3 SET SEXO = 'X';
```

Por último, se vuelve a modificar la tabla con la opción MODIFY y se cambia la definición de la columna a NOT NULL.

Eliminamos las columnas SEXO e IMPORTE de la tabla EJEMPLO3, primero se elimina una columna:

```
ALTER TABLE EJEMPLO3 DROP COLUMN SEXO;
```

y luego la otra: `ALTER TABLE EJEMPLO3 DROP COLUMN IMPORTE;`

15 **Añadiendo una restricción CHECK. Añadimos una restricción a la tabla EMPLE indicando que el SALARIO ha de ser > 0:**

```
ALTER TABLE EMPLE ADD CONSTRAINT SALMAYOR CHECK (SALARIO > 0);
```

Añadiendo una restricción UNIQUE. Añadimos la restricción de APELLIDO único a la tabla EMPLE: `ALTER TABLE EMPLE ADD CONSTRAINT APELLIDO_UQ UNIQUE(APELLIDO);`

Añadiendo una restricción NOT NULL. Añadimos la restricción de COMISION no nula a la tabla EMPLE:

```
ALTER TABLE EMPLE MODIFY COMISION CONSTRAINT COMI_NONULA NOT NULL;
ERROR en línea 1:
ORA-02296: no se puede activar (SCOTT.COMI_NONULA) - se han encontrado valores Nulos
```

En este ejemplo, aparece un error debido a que la columna COMISION es nula en muchas filas de la tabla; para añadir la restricción es necesario dar valores a COMISION para todas las filas de la tabla.

Añadiendo una restricción PRIMARY KEY. Añadimos la restricción de clave primaria a la columna EMP_NO de la tabla EMPLE: `ALTER TABLE EMPLE ADD CONSTRAINT PK_EMPLE PRIMARY KEY(EMP_NO);` Añadimos la restricción de clave primaria a la columna DEPT_NO de la tabla DEPART: `ALTER TABLE DEPART ADD CONSTRAINT PK_DEPART PRIMARY KEY(DEPT_NO);`

Añadiendo una restricción FOREIGN KEY. Añadimos la restricción de clave ajena a la columna DEPT_NO de la tabla EMPLE que referencia a la clave primaria de la tabla DEPART: `ALTER TABLE EMPLE ADD CONSTRAINT FK_EMPLE FOREIGN KEY(DEPT_NO) REFERENCES DEPART ON DELETE CASCADE;`

Veamos ahora las restricciones definidas para cada columna de la tabla EMPLE: `SELECT CONSTRAINT_NAME, COLUMN_NAME FROM USER_CONS_COLUMNS WHERE TABLE_NAME = 'EMPLE';`

```
CONSTRAINT_NAME            COLUMN_NAME
---------------            ---------------
 SYS_C005311                  EMP_NO
 SYS_C005312                  DEPT_NO
 APELLIDO_UQ                  APELLIDO
 SALMAYOR                     SALARIO
 PK_EMPLE                     EMP_NO
 FK_EMPLE                     DEPT_NO
```

Eliminamos algunas de las restricciones definidas en la tabla EMPLE:

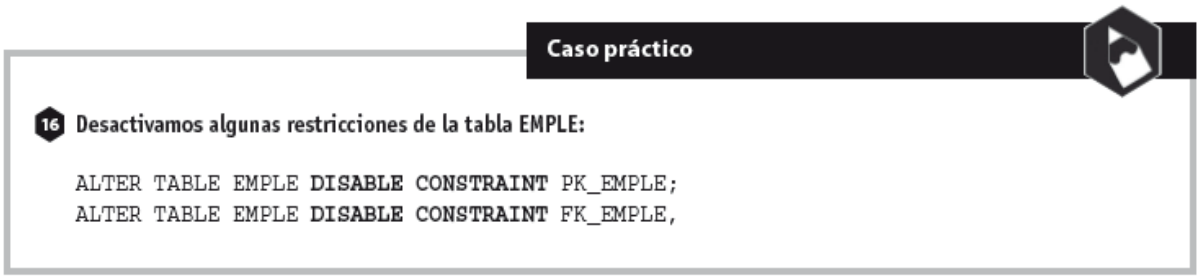
```
ALTER TABLE EMPLE DROP CONSTRAINT SYS_C005311;
ALTER TABLE EMPLE DROP CONSTRAINT APELLIDO_UQ,
```

Activar y desactivar restricciones

Por defecto, las restricciones se activan al crearlas. Se pueden desactivar añadiendo la cláusula DISABLE al final de la restricción.

El siguiente ejemplo añade una restricción, pero la desactiva: ALTER TABLE EMPLE ADD CONSTRAINT APELLIDO_UQ UNIQUE(APELLIDO) **DISABLE**;

Para desactivar una restricción existente usamos la orden:
ALTER TABLE nombretabla **DISABLE CONSTRAINT** nombrerestricción
...
Para activar una restricción existente usamos la orden:
ALTER TABLE nombretabla **ENABLE CONSTRAINT** nombrerestricción
...



CREACIÓN Y USO DE VISTAS.

A veces, para obtener datos de varias tablas hemos de construir una sentencia SELECT compleja y, si en otro momento necesitamos realizar esa misma consulta, tenemos que construir de nuevo la sentencia SELECT.

Sería muy cómodo obtener los datos de una consulta compleja con una simple sentencia SELECT.
Pues bien, las vistas solucionan este problema: mediante una consulta simple de una vista cabe la posibilidad de obtener datos de una consulta compleja.

Una **vista** es una tabla lógica que permite acceder a la información de una o de varias tablas. No contiene información por sí misma, sino que su información está basada en la que contienen otras tablas, llamadas *tablas base*, y siempre refleja los datos de estas tablas; es, simplemente, una sentencia SQL.

Si se suprime una tabla, la vista asociada se invalida. Las vistas tienen la misma estructura que una tabla: filas y columnas, y se tratan de forma semejante a una tabla. El formato para crear una vista es:

CREATE [OR REPLACE] **VIEW** Nombrevista [(columna [,columna])]
AS consulta
[**WITH** {**CHECK OPTION** | **READ ONLY**} **CONSTRAINT** nombrerestricción];

Nombrevista es el nombre que damos a la vista.
[(columna [,columna])] son los nombres de columnas que va a contener la vista.
Si no se ponen, se asumen los nombres de columna devueltos por la consulta.
AS consulta determina las columnas y las tablas que aparecerán en la vista.
[OR REPLACE] crea de nuevo la vista si ya existía.
[**WITH CHECK OPTION**] es la opción de comprobación para una vista. Si se especifica,
SQL comprueba automáticamente cada operación INSERT y UPDATE sobre la vista para asegurarse que las filas resultantes satisfagan el criterio de búsqueda de la definición de la vista.
Si la fila insertada o modificada no satisface la condición de creación de la vista, la sentencia INSERT o UPDATE falla y no se realiza la operación.
[**WITH READ ONLY**] especifica que sólo se puede hacer SELECT de las filas de la vista.
[**CONSTRAINT** nombrerestriccion] especifica el nombre de la restricción WITH
CHECK OPTION o WITH READ ONLY. Es opcional.

Creación y uso de vistas sencillas

Las vistas más sencillas son las que acceden a una única tabla. Por ejemplo, creamos la vista DEP30 que contiene el APELLIDO, el OFICIO y el SALARIO de los empleados de la tabla EMPLE del departamento 30:
CREATE VIEW DEP30 **AS SELECT** APELLIDO, OFICIO, SALARIO FROM EMPLE WHERE DEPT_NO=30;

Ahora la vista creada se puede usar como si de una tabla se tratase. Se puede consultar, se pueden borrar filas, actualizar filas siempre y cuando las columnas a actualizar no sean expresiones (funciones de grupo o referencias a pseudocolumnas); y se puede insertar siempre y cuando todas las columnas obligatorias de la tabla asociada estén presentes
en la vista.
También podríamos haberla creado dando nombre a las columnas, por ejemplo, APE, OFI y SAL:
CREATE OR REPLACE VIEW DEP30 (APE, OFI, SAL)
AS SELECT APELLIDO, OFICIO, SALARIO FROM EMPLE
WHERE DEPT_NO = 30;

Para consultar las vistas creadas se dispone de la vista **USER_VIEWS** y **ALL_VIEWS**. Así, para visualizar los nombres de vistas con sus textos, tenemos:

SQL> SELECT VIEW_NAME, TEXT FROM USER_VIEWS;

Creación y uso de vistas con WITH CHECK OPTION y READ ONLY

Se puede crear una vista de forma que todas las operaciones INSERT, UPDATE y DELETE
que se hagan sobre ella satisfagan la condición por la que se creó. Por ejemplo, creo una vista que contiene todos los datos de los empleados del departamento 20:

CREATE OR REPLACE VIEW DEP20 **AS SELECT** * FROM EMPLE WHERE DEPT_NO = 20;

Ahora inserto en la vista un empleado del departamento 30:

INSERT INTO DEP20 VALUES (3333, 'PEREZ', 'EMPLEADO',7902, SYSDATE, 1300, NULL, 30);

La inserción se realizará correctamente.

Ahora creamos la vista con WITH CHECK OPTION:

CREATE OR REPLACE VIEW DEP20
AS SELECT * FROM EMPLE WHERE DEPT_NO = 20 **WITH CHECK OPTION;**
Intentamos insertar una fila en la vista con el departamento 30; en este caso se producirá un error:

ERROR en línea 1:
ORA-01402: violación de la cláusula WHERE en la vista WITH CHECK OPTION.

La opción **WITH READ ONLY** sólo nos permitirá hacer SELECT en la vista:

CREATE OR REPLACE VIEW DEP30
AS SELECT * FROM EMPLE WHERE DEPT_NO=30 **WITH READ ONLY**;
Cualquier operación INSERT, UPDATE o DELETE sobre la vista DEP30 fallará.

Creación y uso de vistas complejas

Las vistas complejas contienen consultas que se definen sobre más de una tabla, agrupan filas usando las cláusulas GROUP BY o DISTINCT, y contienen llamadas a funciones.

Se pueden crear vistas usando funciones, expresiones en columnas y consultas avanzadas, pero únicamente se podrán consultar estas vistas.

Caso práctico

17 **A partir de las tablas EMPLE y DEPART creamos una vista que contenga las columnas EMP_NO, APELLIDO, DEPT_NO y DNOMBRE :**

```
CREATE VIEW EMP_DEPT (EMP_NO, APELLIDO, DEPT_NO, DNOMBRE) AS
   SELECT EMP_NO,APELLIDO, EMPLE.DEPT_NO, DNOMBRE
   FROM EMPLE, DEPART WHERE EMPLE.DEPT_NO = DEPART.DEPT_NO;
```

Insertamos una fila en la vista: `INSERT INTO EMP_DEPT VALUES(2222,'SUELA', 20, 'INVESTIGACION');`
Pero se produce un error debido a que la vista se creó a partir de dos tablas.

```
ERROR en línea 1:
ORA-01776: no se puede modificar más de una tabla base a través de una vista de unión
```

Los borrados y las modificaciones también producirán errores.

Creamos una vista llamada PAGOS a partir de las filas de la tabla EMPLE, cuyo departamento sea el 10. Las columnas de la vista se llamarán NOMBRE, SAL_MES, SAL_AN y DEPT_NO. El NOMBRE es la columna APELLIDO, a la que aplicamos la función INITCAP( ). SAL_MES es el SALARIO. SAL_AN es el SALARIO*12:

```
CREATE VIEW PAGOS (NOMBRE, SAL_MES, SAL_AN, DEPT_NO)
AS SELECT INITCAP(APELLIDO), SALARIO, SALARIO*12, DEPT_NO FROM EMPLE WHERE DEPT_NO = 10;
```

Podemos modificar filas siempre y cuando la columna que se va a modificar no sea la columna expresada en forma de cálculo (SAL_AN) o la que fue creada mediante la función INITCAP(): `UPDATE PAGOS SET SAL_MES = 5000 WHERE NOMBRE = 'Muñoz';`

Borrado de vistas

Es posible borrar las vistas con la orden DROP VIEW, cuyo formato es:
DROP VIEW nombrevista;

Por ejemplo, borramos la vista PAGOS: **DROP VIEW** PAGOS;

CREACIÓN DE SINÓNIMOS

Cuando tenemos acceso a las tablas de otro usuario y deseamos consultarlas es preciso teclear el nombre del usuario propietario antes del nombre de la tabla.

Es decir, si DIEGO tiene acceso a la tabla DEPART de PEDRO y la quiere consultar, tendrá que teclear la siguiente orden para poder hacerlo: SELECT * FROM **PEDRO.DEPART;**

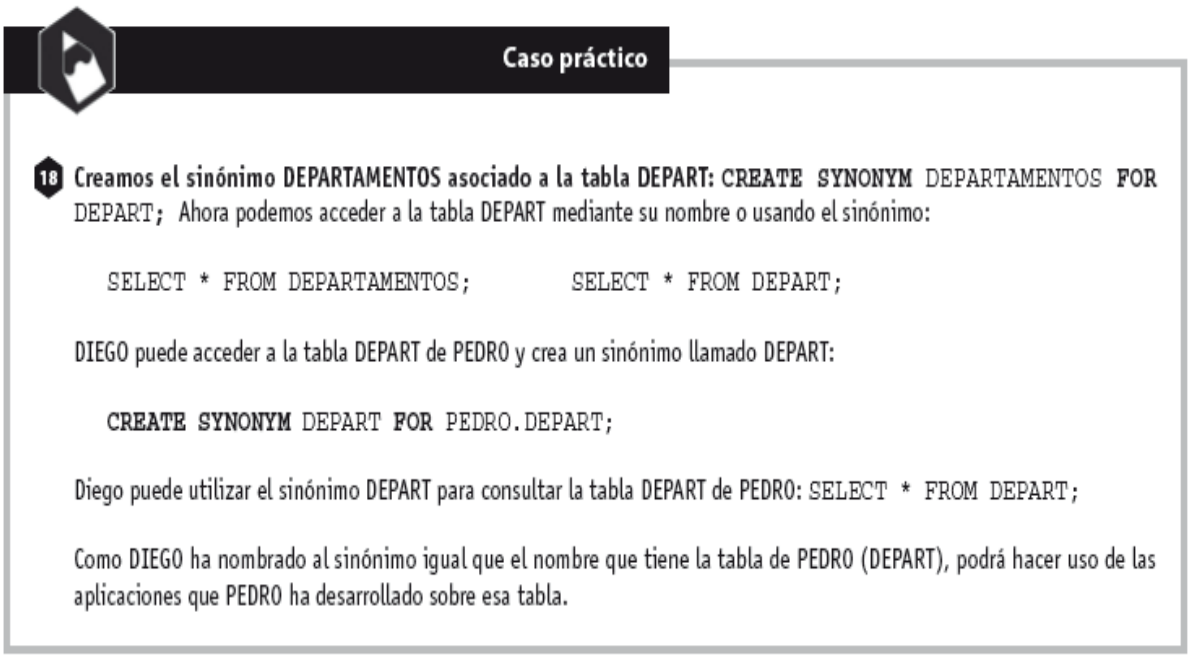
Mediante el uso de sinónimos, DIEGO puede crear un sinónimo para referirse a la tabla de PEDRO sin necesidad de incluir su nombre: SELECT * FROM **TABLAPEDRO;**

Un **sinónimo** es un nuevo nombre que se puede dar a una tabla o vista. Con los sinónimos se pueden utilizar dos nombres diferentes para referirse a un mismo objeto.

Resultan interesantes cuando se tiene acceso a tablas de otros usuarios; se pueden crear sinónimos para hacer referencia a esas tablas y, así, no hay que escribir el nombre de usuario propietario de la tabla delante de la tabla a la que tenemos acceso cada vez que deseemos consultarla.

El formato para crear sinónimos es el siguiente:

CREATE [PUBLIC] **SYNONYM** nombresinónimo **FOR** [usuario.]Nombretabla;
PUBLIC hace que el sinónimo esté disponible para todos los usuarios.



Caso práctico

18 **Creamos el sinónimo DEPARTAMENTOS asociado a la tabla DEPART: CREATE SYNONYM** DEPARTAMENTOS **FOR** DEPART; Ahora podemos acceder a la tabla DEPART mediante su nombre o usando el sinónimo:

```
SELECT * FROM DEPARTAMENTOS;        SELECT * FROM DEPART;
```

DIEGO puede acceder a la tabla DEPART de PEDRO y crea un sinónimo llamado DEPART:

```
CREATE SYNONYM DEPART FOR PEDRO.DEPART;
```

Diego puede utilizar el sinónimo DEPART para consultar la tabla DEPART de PEDRO: `SELECT * FROM DEPART;`

Como DIEGO ha nombrado al sinónimo igual que el nombre que tiene la tabla de PEDRO (DEPART), podrá hacer uso de las aplicaciones que PEDRO ha desarrollado sobre esa tabla.

Por otra parte, existen sinónimos públicos a los que puede hacer referencia cualquier usuario. Sólo el administrador de la base de datos (DBA) y los usuarios con privilegio CREATE PUBLIC SYNONYM pueden crear este tipo de sinónimos. Por ejemplo, un usuario que es DBA crea un sinónimo público para su tabla DEPART; llama DEP al sinónimo:

CREATE PUBLIC SYNONYM DEP FOR DEPART;

Para que todos los usuarios puedan usar la tabla DEPART y su sinónimo han de tener permiso.

Se da permiso a todos los usuarios para hacer SELECT en la tabla DEPART con la orden GRANT: **GRANT SELECT** ON DEPART TO **PUBLIC**; ahora todos los usuarios pueden hacer SELECT del sinónimo público creado para la tabla DEPART: SELECT * FROM **DEP**;

Borrado de sinónimos

Del mismo modo que se crean sinónimos, se pueden borrar, con la orden DROP SYNONYM, cuyo formato es:

DROP [PUBLIC] **SYNONYM** [usuario.]sinónimo;
sinónimo es el nombre de sinónimo que se va a suprimir. Únicamente los DBA y los usuarios con el privilegio DROP PUBLIC SYNONYM pueden suprimir sinónimos PUBLIC.

Igualmente, sólo los DBA y los usuarios con el privilegio DROP ANY SYNONYM pueden borrar los sinónimos de otros usuarios. Por ejemplo, borramos el sinónimo DEPARTAMENTOS:

DROP SYNONYM DEPARTAMENTOS;

Borramos el sinónimo público DEP: **DROP PUBLIC SYNONYM** DEP;
Por otro lado, la vista USER_SYNONYMS permite ver los sinónimos que son propiedad del usuario. Para ver los sinónimos creados por el usuario sobre sus objetos: SELECT SYNONYM_NAME, TABLE_NAME FROM **USER_SYNONYMS;**

CAMBIOS DE NOMBRE

RENAME es una orden SQL que cambia el nombre de una tabla, vista o sinónimo. El nuevo nombre no puede ser una palabra reservada ni el nombre de un objeto que tenga creado el usuario. El formato es éste:
RENAME nombreanterior **TO** nombrenuevo;

Las restricciones de integridad, los índices y los permisos dados al objeto se transfieren automáticamente al nuevo objeto. Oracle invalida todos los objetos que dependen del objeto renombrado, como las vistas, los sinónimos y los procedimientos almacenados que hacen referencia a la tabla renombrada.

No se puede usar esta orden para renombrar sinónimos públicos ni para renombrar columnas de una tabla. Las columnas de una tabla se renombran mediante la orden CREATE TABLE AS.

Conceptos básicos

A continuación se muestra un resumen sobre la orden CREATE TABLE. El formato más básico es el siguiente:

```
CREATE  TABLE  Nombretabla
(
  Columna1   Tipo_dato  [NOT NULL],
  Columna2   Tipo_dato  [NOT NULL],
  ...
) [TABLESPACE espacio_de_tabla];
```

Restricciones en la orden CREATE TABLE:

- Restricción de un solo campo:

```
CONSTRAINT nombrerestricción {
    [NOT] NULL | {PRIMARY KEY | UNIQUE} |
    REFERENCES Nombretabla [(columna[, columna…])]  [ON DELETE CASCADE] |
    CHECK (condición)
}
```

- Restricción de múltiples campos:

```
CONSTRAINT nombrerestricción {
     PRIMARY KEY (columna[, columna …]) |
     UNIQUE (columna[, columna …]) |
     FOREIGN KEY (columna[, columna …])
     REFERENCES Nombretabla [(columna[, columna …])] [ON DELETE CASCADE] |
     CHECK (condición)
}
```

VISTAS DEL DICCIONARIO DE DATOS:

Información de tablas y otros objetos	USER_TABLES, USER_OBJECTS, USER_CATALOG
Información de restricciones	USER_CONSTRAINTS, ALL_CONSTRAINTS, DBA_CONSTRAINTS USER_CONS_COLUMNS, ALL_CONS_COLUMNS, DBA_CONS_COLUMNS
Información sobre vistas	USER_VIEWS, ALL_VIEWS
Información sobre sinónimos	USER_SYNONYMS, ALL_SYNONYMS