



Desarrollo de Aplicaciones Informáticas

CICLO FORMATIVO DE GRADO SUPERIOR

FORMACIÓN PROFESIONAL A DISTANCIA

Unidad

4

SQL – Lenguajes LDD y LCD

MÓDULO

Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE



FORMACIÓN PROFESIONAL

Principado de Asturias

Título del Ciclo: DESARROLLO DE APLICACIONES INFORMATICAS

Título del Módulo: DESARROLLO DE APLICACIONES EN ENTORNOS DE CUARTA GENERACIÓN Y CON HERRAMIENTAS CASE

Unidad 4: **SQL - Lenguajes LDD y LCD**

Dirección: Dirección General de Políticas Educativas, Ordenación Académica y Formación Profesional
Servicio de Formación Profesional Inicial y Aprendizaje Permanente

Dirección de la obra:

Alfonso Careaga Herrera
Antonio Reguera García
Arturo García Fernández
Ascensión Solís Fernández
Juan Carlos Quirós Quirós
Luís M^a Palacio Junquera
Yolanda Álvarez Granda

Coordinador de los contenidos:

Juan Manuel Fernández Gutiérrez

Autores:

Juan Manuel Fernández Gutiérrez
Rodrigo Fernández Martínez

Colección:

Materiales didácticos de aula

Serie:

Formación profesional específica

Edita:

Consejería de Educación y Ciencia

ISBN:

978-84-692-3443-2

Deposito Legal:

AS-3564-2009

Copyright:

2009. Consejería de Educación y Ciencia

Esta publicación tiene fines exclusivamente educativos. Queda prohibida la venta, así como la reproducción total o parcial de sus contenidos sin autorización expresa de los autores y del Copyright

Introducción

SQL es un lenguaje que nos **permite interactuar con los SGBD Relacionales** para especificar las operaciones que deseamos realizar sobre los datos y su estructura. En la unidad anterior hemos utilizado SQL para manipular la información contenida en las tablas.

En esta unidad se explican los fundamentos del SQL estándar para definir las estructuras, modificarlas y establecer los sistemas de seguridad. En definitiva vamos trabajar con los lenguajes de definición y control. Por último veremos funciones avanzadas del lenguaje.

Objetivos

- Conocer las sentencias de definición (LDD)
- Conocer las sentencias de control (LCD)
- Modificar las estructuras de los objetos creados
- Crear y manipular vistas, sinónimos e índices
- Trabajar con múltiples tablas.

Contenidos Generales

| | |
|---|----|
| 1. CREACIÓN Y ELIMINACIÓN DE TABLAS | 3 |
| 1.1. Creación de tablas..... | 3 |
| 1.2. Modificar la definición de las tablas (ALTER)..... | 11 |
| 1.3. Indices | 13 |
| 1.4. Sinónimos | 13 |
| 1.5. Eliminar tablas, índices, sinónimos. | 14 |
| 2. TRANSACCIONES..... | 15 |
| 3. VISTAS..... | 16 |
| 3.1. Vistas simples. | 16 |
| 3.2. Vistas agrupadas. | 17 |
| 3.3. Vistas compuestas..... | 18 |
| 3.4. Acceso a las vistas..... | 18 |
| 3.5. Actualización de vistas. | 19 |
| 3.6. Clausulas de CREATE VIEW..... | 21 |
| 3.7. Eliminar vistas..... | 24 |
| 3.8. Ventajas e inconvenientes de las vistas. | 25 |
| 4. OTRAS FUNCIONES | 25 |
| 4.1. Funciones de conversión y tratamiento de fechas y números..... | 25 |
| 4.2. Resto de Funciones..... | 33 |
| 5. CONFIDENCIALIDAD DE LOS DATOS..... | 37 |
| 5.1 Privilegios y roles..... | 38 |
| 5.2. Autenticación de usuarios. | 42 |
| 6. DESCRIPCIÓN DE LAS TABLAS UTILIZADAS EN ESTAS UNIDADES..... | 44 |

1. Creación y eliminación de tablas

Las sentencias descritas hasta este apartado formaban parte del lenguaje de manipulación de datos. La creación de tablas, modificación de su estructura, eliminación, etc., se llevan a cabo con un conjunto de sentencias que constituyen lo que se llama el **lenguaje de definición de datos (LDD)**.

1.1. Creación de tablas

La creación de tablas no es responsabilidad de los desarrolladores, ya que generalmente son los administradores quienes crean los objetos de la base de datos (tablas, vistas,), sin embargo consideramos que los programadores deben conocer los diferentes tipos de sentencias del lenguaje de definición de datos: crear tablas, vistas, índices, etc. En este caso los usuarios crearán sus propias tablas, serán tablas privadas que cargarán y actualizarán y sobre las que concederán derechos a los usuarios que consideren oportuno, para que puedan acceder a ellas.

Para que un usuario cree tablas en su propio esquema, o en otro tiene que tener los privilegios necesarios.

Para crear las tablas se utiliza la sentencia CREATE que en su formato más simple, sin tener en cuenta las reglas de integridad ni los parámetros de almacenamiento (asume los de por defecto) y siendo el usuario que la crea el propietario, es el siguiente:

```
CREATE TABLE nombre_de_la_tabla  
(  
columna tipo  
[,columna tipo]  
[,columna tipo]  
.....  
);
```

Los tipos de datos son los descritos al principio de la unidad, las columnas se separan con comas y van encerradas entre paréntesis. Si se utilizan nombres compuestos se unen por el carácter subrayado _

La siguiente definición crea la tabla de *cursos*

```
CREATE TABLE CURSOS
(
  CODIGO                VARCHAR2(10) ,
  NOMBRE                VARCHAR2(100) ,
  FECHA_INICIO         DATE ,
  FECHA_FIN             DATE ,
  VACANTES              NUMBER(2) ,
  MODALIDAD             VARCHAR2(100) ,
  OBSERVACIONES        VARCHAR2(400)
);
```

la tabla es propiedad del usuario que la creó, para manipularla sólo tenemos que hacer referencia al nombre de la tabla.

Hasta ahora todas las manipulaciones de tablas las hemos hecho suponiendo que éramos propietarios de las mismas, es decir, que están en nuestro esquema, de no ser así, lo mismo que calificábamos una columna para romper la ambigüedad tendríamos que calificar la tabla para asociarla al esquema al que pertenece.

En lo sucesivo suponemos que somos propietarios de nuestras tablas, por lo tanto no las calificaremos.

Para más información sobre **esquemas** y **usuarios** consultar el epígrafe: Confidencialidad

En las tablas creadas hasta ahora no se controlaba la duplicidad de filas, la relación entre tablas, ni los chequeos de validez, en definitiva, no existían restricciones.

Con las reglas de integridad, tanto de integridad de entidad, como referencial, que se estudiaron en las unidades anteriores, se dota a las tablas de las restricciones necesarias para que los datos almacenados sean consistentes.

A continuación, utilizando la tabla de *alumnos* como modelo, vamos a definir las reglas de integridad.

Como primer paso se trata de encontrar la clave primaria (PRIMARY KEY), que puede ser simple o compuesta y en la que ninguno de sus componentes puede ser nulo, (integridad de entidad). En nuestro caso va ser el *nif*

A continuación se pueden definir claves alternativas (UNIQUE), que son las claves candidatas excepto la elegida como primaria.

Si se considera oportuno, se pueden asignar valores por defecto (DEFAULT) a determinadas columnas. En las altas (inserciones) las columnas que no se les asigna un valor asumen el valor por defecto especificado con DEFAULT. En el caso de que no se les asigne ningún valor y tampoco se les de un valor por defecto asumen nulos.

Podemos establecer chequeos de validez (CHECK). La restricción CHECK permite controlar el valor que se asigna a una columna, comparando el valor asignado con una serie de valores posibles especificados en la CHECK, en caso de no coincidir con ninguno no permite que se grabe la fila. Es aconsejable que el chequeo se realice en la aplicación y no en la base de datos.

También se puede establecer que columnas no pueden ser nulas (NOT NULL).

Por último definiremos la integridad referencial definiendo claves ajenas.

Las definiciones de restricciones de integridad de entidad y de integridad referencial las haremos con la cláusula CONSTRAINT. En Oracle una clave ajena puede hacer referencia a cualquier clave candidata, sea primaria o alternativa.

Tanto si se hace referencia a claves primarias o alternativas, las columnas de referencia deben coincidir en número, tipo y tamaño con las columnas referidas.

Veamos un ejemplo con las definiciones de constraints:

```
CREATE TABLE CURSOS
(
  CODIGO                VARCHAR2(10),
  NOMBRE                VARCHAR2(100),
  FECHA_INICIO         DATE,
  FECHA_FIN            DATE,
  VACANTES             NUMBER(2),
  MODALIDAD            VARCHAR2(100),
  OBSERVACIONES       VARCHAR2(400),
  CONSTRAINT PK_CURSOS PRIMARY KEY(CODIGO)
);
```

Los nombres de las **constraints** (PK_CURSOS) son los nombres con los que se almacenan en el diccionario de datos. Estas **constraints** se denominan de tabla por estar definidas a nivel de tabla (después de definir todas las columnas), se pueden definir a nivel de columna siempre que no intervenga más de una columna en su composición.

```
CREATE TABLE CURSOS
(
  CODIGO                VARCHAR2(10) NOT NULL,
  NOMBRE                VARCHAR2(100),
  FECHA_INICIO         DATE DEFAULT SYSDATE,
  FECHA_FIN            DATE,
  VACANTES             NUMBER(2),
  MODALIDAD            VARCHAR2(100),
  OBSERVACIONES       VARCHAR2(400),
  CONSTRAINT PK_CURSOS PRIMARY KEY(CODIGO)
);
```

```
CREATE TABLE ALUMNOS
(
  MATRICULA            VARCHAR2(10) NOT NULL,
  NOMBRE               VARCHAR2(15),
  APELLIDOS           VARCHAR2(30),
  NIF                  VARCHAR2(10),
  TELEFONO             NUMBER(10),
  E_MAIL              VARCHAR2(100),
```


Unidad 4: SQL Lenguajes LDD y LCD

```
CURSO                                VARCHAR2(10),
CONSTRAINT UQ_CLAVE                  UNIQUE (NIF),
CONSTRAINT PK_ALUMNOS PRIMARY KEY(MATRICULA),
CONSTRAINT FK_CURSOS FOREIGN KEY (CURSO)
REFERENCES CURSOS(CODIGO) ON DELETE CASCADE
);
```

Como norma, utilizaremos CONSTRAINT de tabla para declarar claves ajenas y candidatas

Oracle permite realizar la operación de borrado en cascada.

Con el borrado en cascada, ON DELETE CASCADE, al eliminar una fila de la tabla maestra se borran todas las filas de la tabla relacionada cuyo valor de clave ajena coincida con el valor de la clave primaria borrada.

Con la definición de tablas anteriores si se elimina un curso se borran todos los alumnos que pertenecen a ese curso.

Para crear las tablas os conectáis como *rasty/rasty* y desde **comandos sql** las podéis crear, tal cual se muestra en la siguiente figura

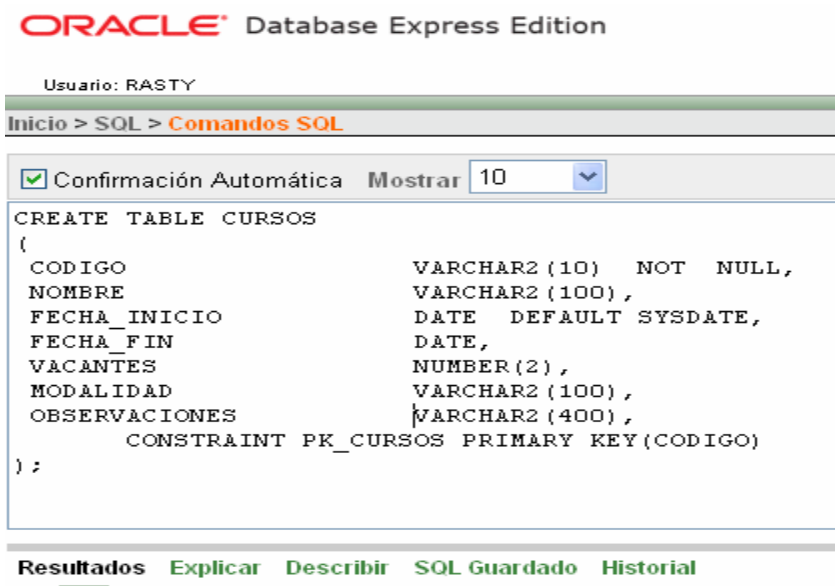
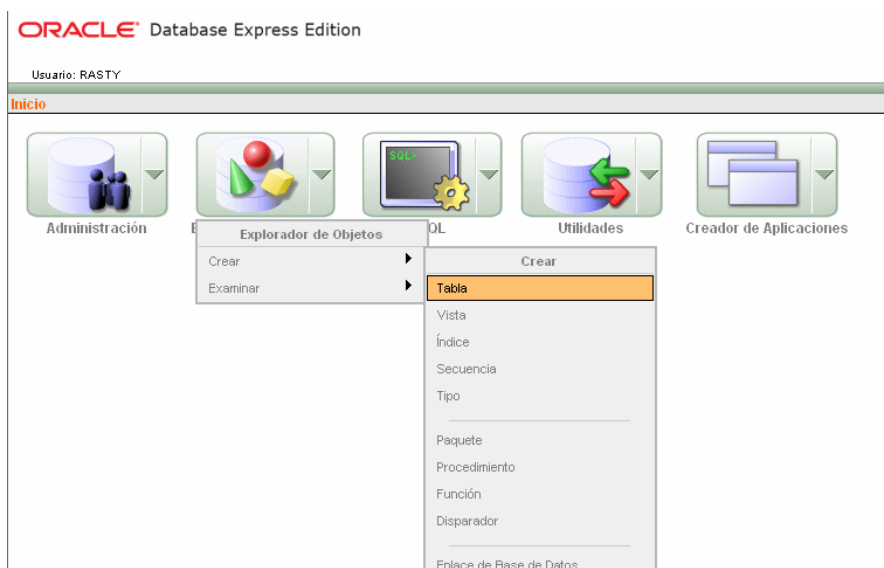


Tabla creada.

También se pueden crear las tablas utilizando específicamente el asistente de Oracle para esta labor. En las secuencias siguientes se muestra como se crea la tabla ALUMNOS relacionada con la de curso por medio de la clave ajena **curso**

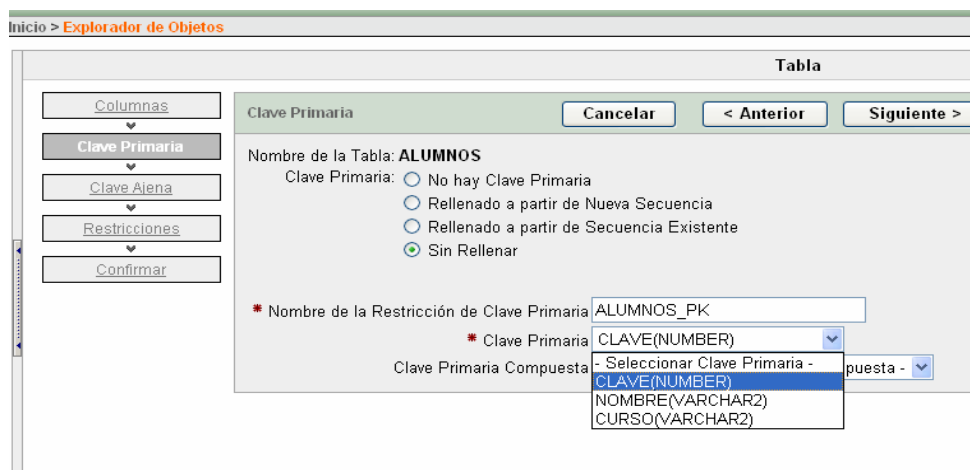
Explorador de objetos → crear → tabla



Se van creando las columnas, dándoles nombre, naturaleza y longitud

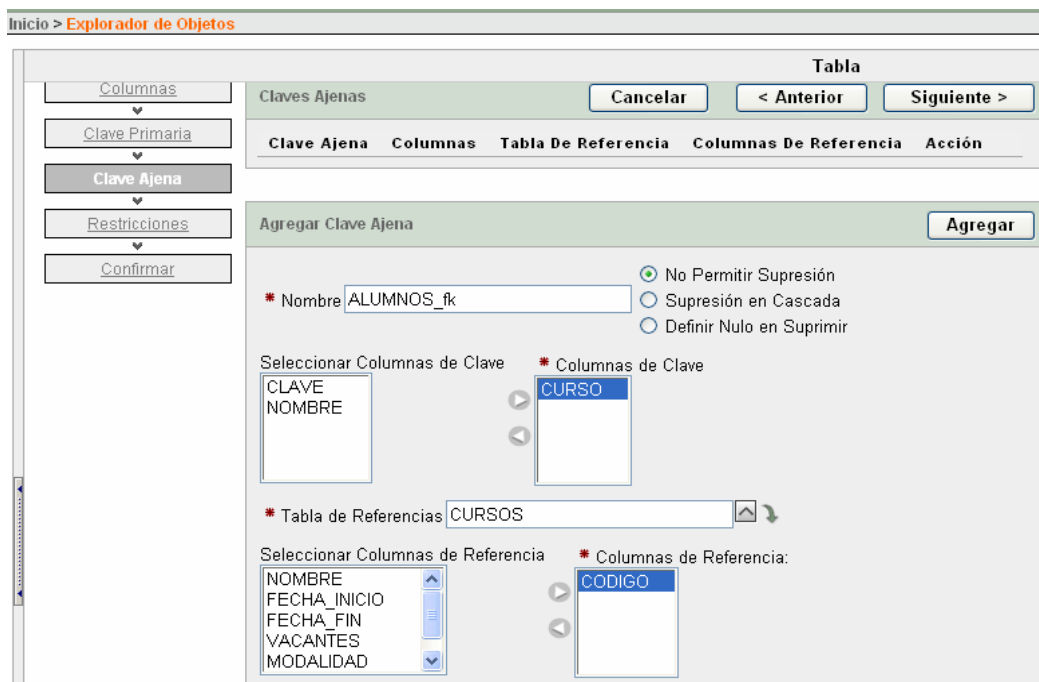


Se define la clave primaria, elegimos la columna CLAVE como primaria. El nombre que propone Oracle para guardar en el catálogo (ALUMNOS_PK) no hace falta cambiarlo. Todos los objetos, incluidas las claves tienen un nombre en el diccionario de datos.



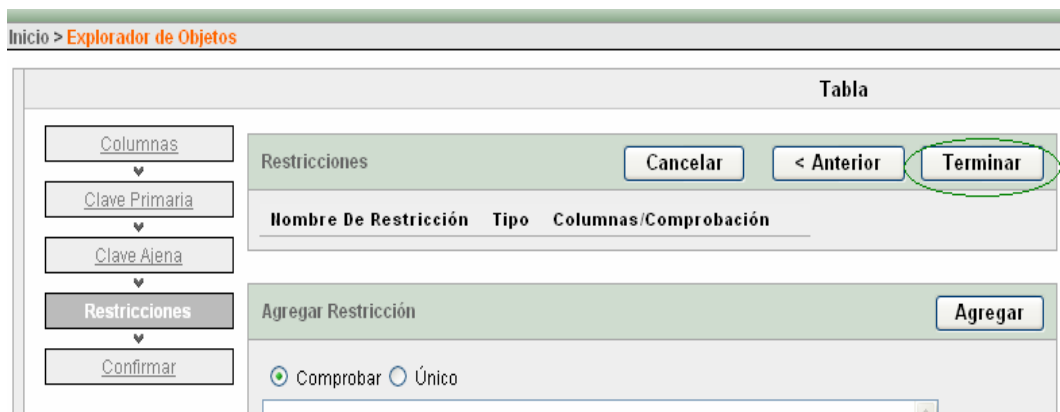
En la siguiente captura se define la clave ajena, en este caso la columna CURSO, que es la columna que va enlazar con la tabla cursos. En tabla de referencias se elige CURSOS y en columnas de referencia se selecciona CODIGO que es la clave primaria de la tabla CURSOS.

Por defecto no se permite supresión, lo que significa que no se pueden eliminar cursos si tiene alumnos relacionados. Si se elige supresión en cascada, permite eliminar el curso y al mismo tiempo elimina los alumnos que relacionados con el curso. Por último la opción de puesta a nulos, permite borrar el curso y poner nulos en la clave ajena CURSO de los alumnos relacionados con el curso que se elimina.

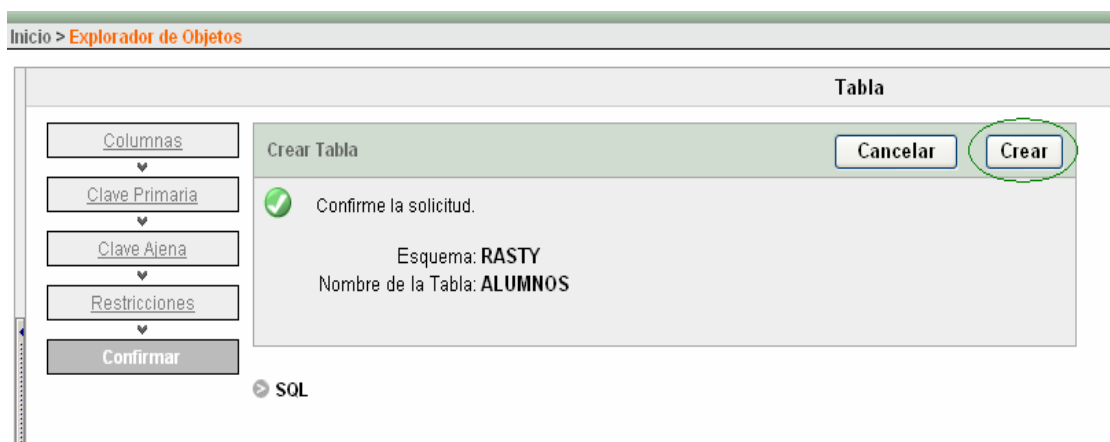


En restricciones no añadimos ninguna. Sería para poner otras restricciones aparte de las claves. Por ejemplo, no permitir nulos, que en una columna se almacene una determinada información (en el caso de que hubiera un PAGA_BANCO que permita almacenar S o N).

Seleccionar la opción TERMINAR



Por último seleccionar CREAR



La siguiente captura muestra la tabla de ALUMNOS



1.2. Modificar la definición de las tablas (ALTER)

Con la sentencia ALTER podemos añadir columnas, nuevas restricciones, redefinir los tipos de datos, tamaños y valores por defecto. También se pueden modificar las características de almacenamiento y desactivar y activar los constraints.

Sintaxis simple de la sentencia ALTER

ALTER TABLE tabla {ADD | MODIFY} {columnas | constraint};

Con ADD añadimos columnas o constraints. Con MODIFY modificamos la definición de las columnas o las constraints.

Ejemplos.

Si queremos añadir a la tabla *dept* la columna *observaciones* emplearemos el siguiente formato de la sentencia ALTER

ALTER TABLE dept ADD (observaciones VARCHAR2(50));

El contenido de la nueva columna es NULL, si queremos definirla como NOT NULL la tabla tiene que estar vacía.

Para modificar el tamaño o el tipo de un dato empleamos la opción MODIFY. En el siguiente ejemplo se modifica el tamaño de la columna *observaciones*

ALTER TABLE dept MODIFY (observaciones VARCHAR2(80));

Sólo se puede cambiar el tipo o decrecer el tamaño de las columnas, si tienen nulos en todas las filas. Por el contrario, se puede incrementar el tamaño con datos en las columnas.

Se pueden definir nuevos valores por defecto, pero la columnas toman esos valores para las nuevas filas que se inserten, las que ya existen permanecen con el valor que tienen.

En cuanto a los controles de integridad, con la cláusula MODIFY sólo se puede añadir NOT NULL a una columna existente, pero con la clausula ADD se pueden añadir todas las constraints (PRIMARY, FOREIGN, UNIQUE,.....)

Suponiendo que en la tabla ALUMNOS no se hubiese definido la alternativa del NIF, se podría hacer con ALTER

ALTER TABLE alumnos ADD CONSTRAINT clave_alternativa_nif UNIQUE (nif);

Para desactivar las constraints utilizaremos la cláusula DISABLE

ALTER TABLE alumnos DISABLE CONSTRAINT clave_alternativa_nif

Para activar las constraints utilizaremos la cláusula ENABLE

ALTER TABLE alumnos ENABLE CONSTRAINT clave_alternativa_nif

1.3. Indices

Podemos indexar la tabla para aquellas columnas que se necesiten accesos rápidos. Se puede indexar una columna simple o una concatenación de varias. Además de los índices que nosotros creamos, el propio S.G.B.D. crea un índice automáticamente cuando se establecen CONSTRAINTS de tipo PRIMARY KEY o UNIQUE

Sintaxis para crear índices

**CREATE INDEX nombre_del_índice
ON nombre_de_la_tabla (columnas [ASC | DDESC]);**

CREATE INDEX i_ape_nom ON alumnos (apellidos, nombre);

El nombre del índice crea una entrada en el diccionario de datos. El número de columnas que se puede indexar por tabla dependerá de la base de datos con la que estemos trabajando. Los nulos no son indexados.

Los índices aceleran las búsquedas pero hacen más lentas las actualizaciones, ya que ellos mismos deben actualizarse.

1.4. Sinónimos .

Los sinónimos son nombres alternativos que se pueden emplear para gestionar las tablas, vistas, secuencias, procedimientos, funciones, paquetes, snapshot, y otros sinónimos.

Formato para crear sinónimos

**CREATE [PUBLIC] SYNONYM nombre_del_sinónimo
FOR {tabla | vista | función | procedimiento |secuencia | paquete |snapshot |sinónimo};**



Los sinónimos pueden ser utilizados en las sentencias de manipulación INSERT, UPDATE, DELETE, SELECT ya comentadas y por las de control GRANT y REVOKE.

Los sinónimos proporcionan independencia para tratar los objetos (tablas, vistas, procedimientos, etc) ya que podemos acceder a objetos de otros propietarios o bases de datos remotas sin necesidad de especificar el propietario ya que va implícito en el nombre del sinónimo.

Suponiendo que el propietario de la tabla *emp* sea *ana*, para hacer una consulta sobre dicha tabla desde otro usuario, que tenga los correspondientes derechos, utilizaríamos la siguiente SELECT

```
SELECT * FROM ana.emp;
```

Si creamos el siguiente sinónimo

```
CREATE SYNONYM con_emp FOR ana.emp;
```

Podemos realizar la consulta de la siguiente forma

```
SELECT * FROM con_emp;
```

1.5. Eliminar tablas, índices, sinónimos.

Para eliminar objetos tales como: tablas, índices, sinónimos. Y otros que se verán en próximas unidades como: procedimientos, funciones, paquetes, disparadores, etc. Se utiliza la sentencia DROP. Cuando se utiliza esta sentencia, la definición de los objetos se pierde junto con su contenido, y no hay forma de recuperarlo. En el caso de las tablas para crear de nuevo la estructura tenemos que volver a utilizar la sentencia CREATE.

```
DROP TABLE alumnos;
```


Con la sentencia anterior eliminamos la tabla de *alumnos*.

Para eliminar una tabla relacionada (con constraints de relación), se empleará el siguiente formato

DROP TABLE alumnos CASCADE CONSTRAINTS;

Para borrar el índice que hemos creado en el ejemplo anterior, empleamos la sentencia:

DROP INDEX i_ape_nom;

Y para borra el sinónimo *con_emp*;

DROP SYNONYM con_emp;

2. Transacciones.

Una transacción es una secuencia de instrucciones SQL que el S.G.B.D gestiona como una unidad de tratamiento. (Más información en la unidad de trabajo PL/SQL)

Una transacción comienza en la primera sentencia SQL, después de una sentencia COMMIT, sentencia ROLLBACK o conexión a la base de datos.

Una transacción termina con una sentencia COMMIT o ROLLBACK o tras una desconexión, intencionada o no, de la base de datos. El S.G.B.D. realiza un COMMIT implícito antes de ejecutar cualquier sentencia DDL (*create, alter,..*) o al realizar una desconexión que no haya sido precedida de un error.

Los cambios realizados a la base de datos en el transcurso de una transacción solo son visibles para el usuario que los ejecuta. Al ejecutar COMMIT los cambios realizados a la base de datos pasan a ser permanentes y por lo tanto visibles para todos los usuarios.

Si una transacción termina con ROLLBACK, se deshacen los cambios realizados a la base de datos por las sentencias previas, quedando en el estado previo al inicio de la transacción.

Durante la ejecución de una transacción, el S.G.B.D. guarda temporalmente información que permita, si se precisa, hacer ROLLBACK y que haga posible realizar lecturas consistentes de la información involucrada en esa transacción. En el caso de Oracle, esta información se almacena en los denominados **segmentos de rollback**

Las sentencias COMMIT y ROLLBACK serán tratadas con más profundidad en la unidad de trabajo PL/SQL

3. Vistas.

Las vistas definen una tabla virtual basada en una o más tablas o vistas. Esta tabla virtual se almacena permanentemente en la base de datos, generando, al igual que las tablas, una entrada en el diccionario de datos. Las vistas permiten organizar los datos de diferentes formas, de modo que los usuarios los ven desde diferentes perspectivas. De igual modo, se restringe el acceso a los datos, permitiendo que determinados usuarios a través de las vistas sólo puedan acceder a determinadas filas y columnas de las tablas.

Desde el punto de vista del usuario, la vista es como una tabla real, con estructura de filas y columnas; pero a diferencia de éstas sus datos no se almacenan físicamente en la base de datos. Las filas y columnas visibles para el usuario son consecuencia de la consulta que se realiza en las tablas reales sobre las que se definió la vista.

La consulta que define la vista permite los operadores relacionales y los de conjunto que se han estudiado en los apartados anteriores. Así podemos aplicar a las vistas los operadores de restricción, proyección, reunión, división. También podemos utilizar los operadores tradicionales de conjuntos: unión, intersección y diferencia.

3.1. Vistas simples.

Para crear una vista utilizamos la sentencia CREATE VIEW, que en su formato más simple tiene la siguiente estructura:

```
CREATE VIEW nombre_vista AS consulta
```

Se puede hacer una clasificación de las vistas atendiendo al tipo de consulta. Si la vista se crea a partir de determinadas filas de la tabla fuente pero con todas sus columnas, obtenemos una vista por restricción.

```
CREATE VIEW sin_comm AS SELECT * FROM emp WHERE comm IS NULL;
```

Con la sentencia anterior se crea una vista con todos los empleados que no tienen comisión

Si construimos la vista con todas o parte de las filas pero con algunas de las columnas de la tabla fuente tendríamos una vista por proyección.

```
CREATE VIEW sin_comm AS SELECT empno, ename FROM emp WHERE comm IS NULL;
```

3.2. Vistas agrupadas.

Otra construcción posible es la que procede de agrupaciones de filas de la tabla original GROUP BY, se denomina vista agrupada, ya que sus filas están construidas por agrupaciones de la tabla fuente. En este caso hay que especificar las columnas que se tienen que corresponder con las de la consulta.

Sintaxis para las vistas agrupadas

```
CREATE VIEW nombre_vista (columnas...) AS consulta con GROUP BY
```

Vamos a crear una vista con los salarios de los empleados agrupados por nombre

Ejemplo

```
CREATE VIEW suma_sal (nombre, salario) AS SELECT ename, sum(sal) FROM  
emp WHERE comm IS NOT NULL GROUP BY ename;
```

La vista anterior estará constituida por los nombres y los salarios de los empleados que tienen comisión.

Al crear una vista podemos dejar que herede los nombres de las columnas de las expresiones que aparecen detrás de la SELECT o darles nosotros nombre, como en este caso que los llamamos *nombre* y *salario*

Para consultar una vista empleamos la sentencia select con el mismo formato que si fuera una tabla. La consulta de la vista que se acaba de crear sería.

```
SELECT * FROM suma_sal;
```

| NOMBRE | SALARIO |
|--------|---------|
| ALLEN | 1600 |
| MARTIN | 1250 |
| TURNER | 1500 |
| WARD | 1250 |

3.3. Vistas compuestas.

Las vistas no tienen porque definirse sobre una sola tabla, pueden estarlo sobre varias tablas. En el siguiente ejemplo la vista *emp_dept* tiene dos columnas obtenidas de las tablas *emp* y *dept*

```
CREATE VIEW emp_dept AS SELECT ename, dname FROM emp, dept  
WHERE emp.deptno=dept.deptno AND sal > 2000;
```

En este caso la vista se construye con los nombres de los empleados y departamentos, excluyendo aquellos empleados cuyo salario es inferior a 2000.

3.4. Acceso a las vistas

El usuario accede a los datos de una vista exactamente igual que si estuviese haciéndolo a una tabla. De hecho, en muchos casos el usuario no sabrá siquiera que está consultando una vista. En general, el uso de las vistas simplifica las consultas a la base de datos.

La siguiente consulta se hace sobre la vista creada en el epígrafe anterior.

```
SELECT * FROM emp_dept;
```

| ENAME | DNAME |
|-------|------------|
| JONES | RESEARCH |
| BLAKE | SALES |
| CLARK | ACCOUNTING |
| SCOTT | RESEARCH |
| KING | ACCOUNTING |
| FORD | RESEARCH |

Las consultas actúan sobre las vistas igual que sobre las tablas reales, por lo tanto podemos trabajar con las mismas cláusulas de la SELECT que se describieron en las consultas sobre tablas.

3.5. Actualización de vistas.

Para que una vista se pueda actualizar debe de existir una relación directa entre las filas y las columnas de la vista y las de la tabla fuente. Por ejemplo, en el caso siguiente sí se pueden realizar inserciones, borrados y actualizaciones.

```
CREATE VIEW sin_comm AS SELECT * FROM emp WHERE comm IS NULL;
```

La vista **sin_comm** tiene la misma estructura que la tabla fuente, luego las actualizaciones que se realicen sobre ella se realizan sobre la tabla fuente.

No se pueden actualizar las vistas que para su construcción se hayan incluido en las consultas las cláusulas:

- GROUP BY o HAVING
- En la WHERE subconsultas
- En las columnas que tienen expresiones o funciones
- En la FROM varias tablas
- La cláusula DISTINCT

Estas reglas no son aplicables a todos los Sistemas Gestores de Bases de Datos. Se consideran reglas generales reguladas por ANSI/ISO, pero cada SGBD tiene sus particularidades y por lo tanto se debe estudiar en cada caso.

Insertamos una fila en la vista *sin_comm*

```
INSERT INTO sin_comm (empno,ename,sal) VALUES (9999,'BALDOMERO', 9000);
```

Si ahora seleccionamos las filas de la tabla fuente

```
SELECT empno, ename, sal FROM emp;
```

| EMPNO | ENAME | SAL |
|-------|-----------|------|
| 7369 | SMITH | 800 |
| 7499 | ALLEN | 1600 |
| 7521 | WARD | 1250 |
| 7566 | JONES | 2975 |
| 7654 | MARTIN | 1250 |
| 7698 | BLAKE | 2850 |
| 7782 | CLARK | 2450 |
| 7788 | SCOTT | 3000 |
| 7839 | KING | 5000 |
| 7844 | TURNER | 1500 |
| 7876 | ADAMS | 1100 |
| 7900 | JAMES | 950 |
| 7902 | FORD | 3000 |
| 7934 | MILLER | 1300 |
| 9999 | BALDOMERO | 9000 |

Vemos que insertó la fila a través de la vista.

Si consultamos la vista

| EMPNO | ENAME | SAL |
|-------|-----------|------|
| 7369 | SMITH | 800 |
| 7566 | JONES | 2975 |
| 7698 | BLAKE | 2850 |
| 7782 | CLARK | 2450 |
| 7788 | SCOTT | 3000 |
| 7839 | KING | 5000 |
| 7876 | ADAMS | 1100 |
| 7900 | JAMES | 950 |
| 7902 | FORD | 3000 |
| 7934 | MILLER | 1300 |
| 9999 | BALDOMERO | 9000 |

Se observa que la fila está en la vista

Vamos intentar insertar una fila en la vista *emp_dept*

INSERT INTO emp_dept VALUES ('baldomero', 'contabilidad');

El sistema visualiza el siguiente mensaje:

ERROR en línea 1:

ORA-01776: no se puede modificar más de una tabla base a través de una vista de unión

El error se debe a la imposibilidad de actualizar las tablas fuente cuando la vista se construye con una selección multitable.

Cuando hacemos una inserción en una vista definida sobre una sola tabla pero que no coge todas sus columnas, sucede que:

- Las columnas de la tabla que no están en la vista y tienen la restricción DEFAULT asumen el valor por defecto.
- Las columnas de la tabla que no están en la vista y no tienen la restricción DEFAULT, asumen nulos.
- Si las columnas de la tabla que no están en la vista tienen la restricción NOT NULL y no la DEFAULT no se inserta la fila .

3.6.Clausulas de CREATE VIEW

A continuación exponemos algunas de las opciones que se pueden utilizar cuando se crea una vista .

- **Cláusula CHECK OPTION.**

Acabamos de ver que las vistas son actualizables cuando existe una relación directa entre las filas y columnas de la vista con las de la tabla fuente, sin embargo podemos observar que si insertamos filas con una condición, la tabla fuente se actualiza con dichas filas, pero si la consulta que define la vista no satisface dicha condición para las filas insertadas, éstas no son

visibles a través de la vista. Dicho de otra forma, cuando se define una vista simple por restricción, a través de la vista solo son visibles aquellas filas de la tabla base que cumplen la restricción.

Veamos el contenido de la tabla *emp*:

```
SELECT empno, ename, sal FROM emp;
```

Creemos una vista con los empleados cuyo salario exceda de 2000

```
CREATE VIEW vista_emp AS SELECT empno, ename, sal FROM emp WHERE sal > 2000;
```

A través de ella se visualiza

```
SELECT * FROM vista_emp;
```

| EMPNO | ENAME | SAL |
|-------|-------|------|
| 7566 | JONES | 2975 |
| 7698 | BLAKE | 2850 |
| 7782 | CLARK | 2450 |
| 7788 | SCOTT | 3000 |
| 7839 | KING | 5000 |
| 7902 | FORD | 3000 |

Tal como fue definida esta vista, es posible insertar filas que luego no sean visibles a través de ella misma.

Insertamos una fila con salario inferior a 2000

```
INSERT INTO vista_emp VALUES (9999,'LAURA', 1000);
```

Si ahora realizamos una consulta de la vista comprobamos que la fila insertada no es visible


```
SELECT * FROM vista_emp;
```

| EMPNO | ENAME | SAL |
|-------|-------|------|
| 7566 | JONES | 2975 |
| 7698 | BLAKE | 2850 |
| 7782 | CLARK | 2450 |
| 7788 | SCOTT | 3000 |
| 7839 | KING | 5000 |
| 7902 | FORD | 3000 |

Sin embargo la fila si está en la tabla fuente.

```
SELECT empno, ename, sal FROM emp;
```

| EMPNO | ENAME | SAL |
|-------------|--------------|-------------|
| 7369 | SMITH | 800 |
| 7499 | ALLEN | 1600 |
| 7521 | WARD | 1250 |
| 7566 | JONES | 2975 |
| 7654 | MARTIN | 1250 |
| 7698 | BLAKE | 2850 |
| 7782 | CLARK | 2450 |
| 7788 | SCOTT | 3000 |
| 7839 | KING | 5000 |
| 7844 | TURNER | 1500 |
| 7876 | ADAMS | 1100 |
| 7900 | JAMES | 950 |
| 7902 | FORD | 3000 |
| 7934 | MILLER | 1300 |
| 9999 | LAURA | 1000 |

Con la cláusula CHECK OPTION se asegura que las operaciones de inserción y actualización sobre la vista satisfagan el criterio de búsqueda en la WHERE de la definición de la vista. De esta forma se asegura que las filas actualizadas o insertadas van a ser visibles a través de la vista.

```
CREATE VIEW vista_emp AS SELECT empno, ename, sal FROM emp WHERE  
sal > 2000
```

```
WITH CHECK OPTION;
```

Si intentamos insertar una fila con un salario menor que 2000.

```
INSERT INTO vista_emp VALUES (9999,'LAURA', 1000);
```

Produce el siguiente error.

```
*  
ERROR en línea 1:  
ORA-01402: violación de la cláusula WHERE en la vista WITH CHECK OPTION  
Cláusula OR REPLACE
```

Cuando se necesitan modificar la definición de una vista, la única solución es crear una nueva vista con el mismo nombre, ya que no existe una sentencia parecida a la ALTER TABLE que permitía modificar la definición de una tabla.

- **Cláusula OR REPLACE**

```
CREATE OR REPLACE VIEW vista_emp AS SELECT empno, ename, sal FROM emp;
```

La cláusula OR REPLACE de la sentencia CREATE VIEW permite sustituir la definición de una vista por una nueva definición

- **Cláusula WITH READ ONLY**

No se permiten borrados, inserciones o actualizaciones a través de la vista.

3.7. Eliminar vistas.

Las vistas se eliminan con la sentencia DROP, la cual tiene el mismo formato que el utilizado para eliminar tablas, índices,..

```
DROP VIEW nombre_de_la_vista
```

Si la vista está en otro esquema y tenemos derechos, podemos borrarla anteponiéndole el nombre del usuario

```
DROP VIEW nombre_usuario.nombre_vista
```

3.8. Ventajas e inconvenientes de las vistas.

Las vistas proporcionan una serie de beneficios como:

- La simplicidad de las consultas, haciendo que consultas con selecciones complejas se simplifiquen a través de las vistas.
- La personalización de la base de datos para determinados usuarios, de forma que presenta los datos con una estructura lógica para dichos usuarios.
- Como consecuencia del apartado anterior se controla el acceso a la base de datos a través de las vistas, dejando que el usuario pueda ver y manejar determinada información

Como principales inconvenientes, podemos citar:

- Las restricciones referidas a actualizaciones.
- Caída del rendimiento cuando se construyen vistas con selecciones complejas.

4. Otras funciones

Además de las funciones de cadena y aritméticas descritas, existen otras funciones tales como funciones de conversión, tratamiento de fechas y otras de propósito general. En este apartado vamos describir algunas de estas funciones.

4.1. Funciones de conversión y tratamiento de fechas y números

Este tipo de funciones convierten un tipo de datos en otro. En este apartado describiremos las más utilizadas, en cualquier caso cada implementación de SQL incorpora, además de las que aquí se describen funciones propias de conversión.

Las funciones de conversión y tratamiento de fechas varían dependiendo del S.G.B.D que se este utilizando, en nuestro caso nos vamos ajustar las funciones definidas por ORACLE.

- **Función TO_CHAR**

Función que convierte datos de tipo numérico o fecha a tipo VARCHAR2.

Sintaxis Formato de TO_CHAR para conversión de fechas

TO_CHAR(dato [, mascara ['nlsparams']])

La función TO_CHAR para conversión de fechas, convierte el dato de tipo DATE al tipo VARCHAR2 según el formato especificado por la máscara. Si se omite la máscara el dato es convertido a VARCHAR2 según el formato por defecto.

El parámetro *nlsparams* especifica el lenguaje en el que aparecerán escritos los nombres y abreviaturas de meses y días de la semana. El formato de este parámetro es **NLS_DATE_LANGUAGE = lenguaje**. Si se omite *nlsparams*, esta función utiliza el lenguaje por defecto de la sesión.

Elementos de la mascara para fechas

| | |
|-------------|--|
| Separadores | - / . : ; "texto" |
| MM | número del mes |
| MON | abreviatura del mes |
| MONTH | nombre del mes relleno con espacios hasta una longitud de 9 caracteres |
| RM | mes en números romanos |
| DDD | número del día del año |
| DD | número del día del mes |
| D | número del día de la semana |
| DY | abreviatura del día de la semana |
| DAY | nombre del día de la semana completo |
| YYYY | año con cuatro dígitos |
| YY | año con dos dígitos |
| YEAR | año en letra |
| RR | dos últimos dígitos del año, permitiendo almacenar fechas del siglo 21 en el 20 y viceversa |
| Q | número del trimestre |
| WW | semana del año (1-53) donde la semana 1 comienza en el día primero del año y continua hasta el día séptimo |
| W | semana del mes (1-5) donde la semana 1 comienza el día 1 del mes y termina el día 7 |
| HH | hora del día |
| HH12 | hora del día (1-12) |
| HH24 | hora del día (1-24) |
| MI | minuto de la hora |
| SS | segundo del minuto |
| AM | muestra AM o PM dependiendo de la hora del día |



Hemos descrito los elementos que consideramos más significativos. Consultar el manual o ayuda en línea para ver el resto de elementos que se pueden utilizar en la máscara.

Con TO_CHAR podemos cambiar el formato en edición.

Si editamos la fecha del sistema

SELECT SYSDATE FROM DUAL;

Por defecto produce la siguiente salida (toma la máscara que tiene definida por defecto para la sesión)

```
SYSDATE
-----
01/11/08
```

Si queremos editarla con otra máscara:

```
SELECT TO_CHAR(SYSDATE,'DD-MM-YYYY') FROM DUAL;
```

```
TO_CHAR(SY
-----
11-02-2008
```

Para conocer el día del año:

```
SELECT TO_CHAR(SYSDATE,'YYDDD') FROM DUAL;
```

```
TO_CH
-----
08042
```

El trimestre

```
SELECT TO_CHAR(SYSDATE,'Q') FROM DUAL;
```

```
T
-
1
```

- ***Función TO_CHAR para la conversión de números.***

La función TO_CHAR para la conversión de números tiene el mismo formato que la TO_CHAR para la conversión de fechas, en este caso el dato es de tipo NUMBER. Si se omite la máscara convierte el número a un tipo VARCHAR2 de la longitud necesaria para contener todos sus dígitos significativos.

El parámetro ***nlsparams*** especifica los caracteres que serán utilizados para representar:

Separador de decimales

Separador de millares

Símbolo monetario local

Símbolo monetario internacional

Este argumento puede tener esta forma:

NLS_NUMERIC_CHARACTERS = 'dg'

NLS_CURRENCY = "text"

NLS_ISO_CURRENCY = 'territory'

Los caracteres ***d*** y ***g*** representan el separador de decimales y de millares respectivamente. Deben ser caracteres diferentes. Se pueden usar hasta diez caracteres para el símbolo monetario.

Si se omite ***nlsparams*** o cualquiera de sus componentes, esta función usa los valores por defecto de los parámetros para la sesión.

Elementos de la máscara para edición de números

| Elemento | Ejemplo | Descripción |
|----------|---------|--|
| 9 | 999 | Devuelve el valor numérico eliminando ceros a la izquierda |
| 0 | 00999 | Con ceros por la izquierda |
| \$ | \$999 | Devuelve el valor con el \$ |
| MI | 999MI | Devuelve el signo menos por la derecha si el valor es negativo |
| S | S999 | Devuelve - si es negativo y + si es positivo |
| G | 99G999 | Separador de grupo de enteros |

| | | |
|------|---------|--|
| D | 99D99 | Separador de grupo de decimales |
| , | 99,999 | Puntuación de millar |
| . | 99.99 | Puntuación decimal |
| EEEE | 9.9EEEE | Devuelve el valor en notación científica |

Ejemplos

Consulta del salario de JONES sin máscara de edición

```
SELECT ename, sal FROM emp WHERE ename = 'JONES';
```

```
NAME          SAL
-----
JONES         2975
```

Consulta del salario de JONES con máscara de edición

```
SELECT ename, TO_CHAR(sal,'9G990D00') FROM emp WHERE ename = 'JONES';
```

```
ENAME          TO_CHAR(S
-----
JONES         2.975,00
```

- **Función TO_DATE**

Convierte el dato de tipo CHAR o VARCHAR2 al tipo DATE. La máscara es un formato de fecha que especifica en que forma está representado el dato. Si se omite la máscara, el dato debe de estar en el formato de fecha por defecto.

Sintaxis de TO_DATE

```
TO_DATE(dato [, máscara ['nlsparams']]);
```

Con esta función podemos ajustar la fecha a una determinada máscara.

- **Modelos de formatos de fechas**

Se pueden usar modelos de formatos de fechas en los siguientes lugares:

- En la función TO_CHAR para convertir un valor de tipo DATE que esté en un formato distinto del formato por defecto
- En la función TO_DATE para convertir un valor de tipo carácter que esté en un formato distinto del formato por defecto

El formato de fechas por defecto se especifica explícitamente con el parámetro de inicialización NLS_DATE_FORMAT o implícitamente con el parámetro de inicialización NLS_TERRITORY.

Aquellos elementos de la máscara que devuelven valores en letra (MONTH, DAY, ..) lo harán en el idioma especificado explícitamente con el parámetro de inicialización NLS_DATE_LANUAJE o implícitamente con NLS_LANUAJE.

- **El elemento RR de la máscara.**

Este elemento es similar al YY, pero proporciona una flexibilidad adicional para almacenar valores de fechas en otros siglos. Esta máscara permite almacenar fechas del siglo 21 en el 20 especificando solo los dos últimos dígitos del año. También se puede hacer el proceso inverso, almacenar fechas del 20 en el 21.

Si se usa la función TO_DATE con la máscara YY, el valor devuelto pertenece siempre al siglo actual. Si se usa la máscara de formato RR en su lugar, el siglo devuelto varía con los dos últimos dígitos del año especificado y con los dos últimos del año actual. En la siguiente tabla se especifica el funcionamiento de este elemento.

| | | |
|---|--|---|
| Si los dígitos del año especificado son | | |
| Si los dos últimos Dígitos del año actual Son | 0 - 49 | 50 - 99 |
| 0 - 49 | La fecha devuelta está en el siglo actual | La fecha devuelta está en el siglo anterior al actual |
| 50 - 99 | La fecha devuelta está en el siglo siguiente al actual | La fecha devuelta está en el siglo actual |

Ejemplos:

Las consultas se hacen con la fecha del ordenador en el 2008

Consulta con máscara RR

```
SELECT TO_CHAR(TO_DATE ('27-OCT-95','DD-MON-RR'), 'YYYY') "Año" FROM DUAL;  
  
Año  
----  
1995
```

Consulta con mascara YY

```
SELECT TO_CHAR(TO_DATE ('27-OCT-95','DD-MON-YY'), 'YYYY') "Año" FROM DUAL;  
  
Año  
----  
2095
```

- **Función TO_NUMBER**

Convierte un dato tipo CHAR o VARCHAR2 con contenido numérico, en un valor de tipo NUMBER según el formato especificado por la máscara.

El parámetro *nlsparams* tiene el mismo propósito que en la función TO_CHAR para conversión de números.

TO_NUMBER(dato [, mascara [, 'nlsparams']])

Ejemplo con la función TO_NUMBER:

```
SELECT TO_NUMBER('-008000','S999999') FROM DUAL;
```

```
TO_NUMBER( '-008000' , 'S999999' )
```

```
-----  
-8000
```

- **Función *ADD_MONTHS***

Suma meses a una fecha

```
ADD_MONTHS(fecha, cantidad_meses)
```

Ejemplo.

Obtenemos la fecha del sistema con formato DD-MM-YYYY

```
SELECT TO_CHAR(SYSDATE,'DD-MM-YYYY') "Fecha" FROM DUAL;
```

```
Fecha
```

```
-----  
12-02-2008
```

Sumamos cuatro al mes

```
SELECT TO_CHAR(ADD_MONTHS(SYSDATE,4),'DD-MM-YYYY') "Nueva fecha" FROM DUAL;
```

```
Nueva fecha
```

```
-----  
12-06-2008
```

- **Función MONTHS_BETWEEN**

Devuelve el número de meses entre dos fechas

MONTHS BETWEEN(fecha1. fecha2)

SELECT MONTHS_BETWEEN('01-05-98','01-01-97') FROM DUAL;

```
MONTHS_BETWEEN('01-05-98','01-01-97')
```

```
-----
```

16

- **Función LAST_DAY**

Devuelve la fecha correspondiente al último día del mes de la fecha que se consulta.

LAST_DAY(fecha)

**SELECT SYSDATE, LAST_DAY(SYSDATE) "Ultimo " ,
LAST_DAY(SYSDATE) - SYSDATE "Diferencia" FROM DUAL;**

```
SYSDATE Ultimo Diferencia
```

```
-----
```

```
12/02/08 29/02/08 17
```

4.2. Resto de Funciones.

A continuación describimos algunas funciones que podríamos denominar de propósito general por englobarlas de alguna manera en un único epígrafe. En cualquier caso no se contemplan todas las funciones disponibles. Invitamos al lector que consulte el manual de SQL .

- **Función LEAST**

Selecciona el menor valor de una lista. Los valores pueden ser columnas, literales, expresiones,...

LEAST(VALOR1, VALOR2,)



En el caso de fechas, selecciona la menor fecha de la lista. Si la fecha es un literal hay que convertirla con TO_DATE

Primero vemos cual es la fecha del sistema

```
SELECT TO_CHAR(SYSDATE,'DD-MM-YYYY') "Fecha" FROM DUAL;
```

```
Fecha  
-----  
12-02-2008
```

Luego seleccionamos la más pequeña, entre ella y el 01-01-08

```
SELECT TO_CHAR(LEAST(SYSDATE,'01-01-08'),'DD-MM-YYYY') "LA MENOR"  
FROM DUAL;
```

```
LA MENOR  
-----  
01-01-2008
```

- ***Función GREATEST***

Selecciona el mayor valor de una lista. Los valores pueden ser columnas, literales, expresiones,...

GREATEST (VALOR1, VALOR2,...)

En el caso de fechas, selecciona la mayor fecha de la lista.

Se puede establecer una comparación entre valor fecha con un literal que contenga una fecha con formato por defecto sin necesidad de convertirlo con TO_DATE.

No se pueden comparar literales aunque sean fechas, sin antes convertirlos con TO_DATE

- **Función NVL.**

Se utiliza para sustituir nulos por un valor real.

NVL(expresion1, expresion2)

Si **expresion1** y **expresion2** son tipos diferentes, primero hay que convertir **expresion2** al tipo **expresion1** y luego se compara.

En el ejemplo siguiente se cambian los nulos obtenidos en la consulta por 0. (Las comisiones nulas por cero)

SELECT NVL(COMM,0) FROM EMP;

```
NVL ( COMM , 0 )
-----
      0
     300
     500
      0
    1400
      0
      0
      0
      0
      0
      0
      0
      0
      0
      0
      0
```

- **Función UID**

Devuelve el identificador del usuario. Es de tipo numérico.

SELECT UID FROM DUAL;

```
UID
-----
 41
```

- **Función USER**

Devuelve el nombre del usuario. Es de tipo carácter.

SELECT USER FROM DUAL;

```
USER
-----
SCOTT
```

- **Función USERENV**

Devuelve información de tipo carácter acerca de la sesión que está activa.

USERENV (opciones)

Ejemplo.

Lenguaje definido para la sesión activa.

SELECT USERENV('LANGUAGE') FROM DUAL;

```
USERENV( ' LANGUAGE ' )
-----
SPANISH_SPAIN.WE8MSWIN1252
```

- **Función VSIZE**

Devuelve el número de bytes que ocupan los datos seleccionados.

```
SELECT dname, VSIZE(dname) "BYTES" FROM dept;
```

| DNAME | BYTES |
|------------|-------|
| ACCOUNTING | 10 |
| RESEARCH | 8 |
| SALES | 5 |
| OPERATIONS | 10 |

5. Confidencialidad de los datos.

En este apartado veremos los elementos del lenguaje SQL que facilitan la protección de la base de datos contra accesos no autorizados.

El lenguaje SQL estándar soporta el sistema de gestión de confidencialidad conocido como **control de acceso discrecional**. Este sistema se basa en que cada usuario tendrá diferentes derechos de acceso, también conocidos como **privilegios**, sobre distintos objetos de la base de datos. Normalmente, usuarios diferentes tendrán privilegios diferentes sobre el mismo objeto. El mecanismo de vistas soportado por SQL facilita también la gestión de la confidencialidad, ocultando información a usuarios no autorizados.

Oracle utiliza el concepto de **esquema** o colección con nombre de objetos, como tablas, vistas, procedimientos y packages. Un usuario es un nombre definido en la base de datos con el que se puede conectar y acceder a los objetos en los esquemas de la base de datos.

Cuando se crea un usuario en la base de datos, se crea también un esquema con el mismo nombre para el usuario. Por defecto, cuando el usuario se conecta con la base de datos, tiene acceso a todos los objetos del esquema correspondiente. Un usuario está asociado sólo con el esquema de su mismo nombre.

Disponiendo de los privilegios correspondientes, un usuario puede trabajar en cualquier esquema, no sólo en el suyo. El nombre de cualquier objeto puede calificarse con el nombre del esquema en que se encuentra

[nombre del esquema.] nombre del objeto

Si se omite el nombre del esquema se utiliza el esquema del usuario.

5.1 Privilegios y roles.

Oracle, para facilitar la administración de privilegios, utiliza el concepto de **rol** o “grupo con nombre de privilegios que pueden ser concedidos o revocados a usuarios o a otros roles”. Soporta dos tipos de privilegios: **privilegios de objetos** y **privilegios del sistema**.

- **Privilegios de objeto**

Un **privilegio de objeto** es un derecho para realizar una acción concreta en una tabla, vista, secuencia, procedimiento, función o package específicos. Por ejemplo, el privilegio para borrar filas en la tabla *usuarios* es un privilegio de objeto. Dependiendo del tipo de objeto, hay diferentes tipos de privilegios.

Las sentencias GRANT y REVOKE permiten que un usuario conceda o retire privilegios a otros usuarios sobre los objetos de su propiedad.

El creador de un objeto, tiene automáticamente concedidos todos los privilegios que son aplicables a dicho objeto. Por ejemplo, el creador de la tabla *emp* tiene automáticamente los privilegios SELECT, INSERT, UPDATE, DELETE y REFERENCES en *emp*. Además, tiene concedidos estos privilegios con WITH GRANT OPTION, es decir con autoridad para concederlos a otros usuarios.

Sintaxis de la sentencia GRANT

```
GRANT {lista de privilegios | ALL [PRIVILEGES]} [(columna1, columna2,...)] ON objeto  
TO {lista de usuarios | roles | PUBLIC} [WITH GRANT OPTION ]
```

Donde:

La **lista de privilegios** consiste en uno o más nombres de privilegios separados por comas. (ALTER, DELETE, EXECUTE, INDEX, INSERT, REFERENCES, SELECT, UPDATE)

ALL PRIVILEGES concede todos los privilegios al objeto especificado.

Columna1, columna2,.. especifica las columnas de la tabla o vista sobre las que se conceden los privilegios (solo para INSERT, REFERENCES y UPDATE)

Objeto se refiere al nombre del objeto sobre el que se conceden privilegios. (tablas, vistas, secuencias, procedimientos, funciones, paquetes, sinonimos)

La **lista de usuarios** serán los identificativos de uno o más usuarios separados por comas. Se puede usar PUBLIC, con el significado de todos los usuarios definidos en el sistema.

Si se especifica **WITH GRANT OPTION**, los usuarios a los que se conceden los privilegios podrán a su vez concederlos a otros usuarios.

Un usuario que concede ciertos privilegios a otro, puede posteriormente revocar dicha concesión. La eliminación de privilegios sólo puede ser realizada por aquél que los concedió y para llevar a cabo la revocación de la concesión de uno o mas privilegios se utiliza la sentencia **REVOKE**.

Sintaxis de la sentencia REVOKE

```
REVOKE {lista de privilegios | ALL [PRIVILEGES] } ON objeto  
FROM {lista de usuarios | roles | PUBLIC} [ CASCADE CONSTRAINTS ]
```

- **Lista de privilegios, objeto, lista de usuarios** tienen el mismo significado que para GRANT
- **CASCADE CONSTRAINTS**, cuyo significado se explica a continuación.

Supongamos que el usuario **A** concede el privilegio **p** sobre cierto objeto al usuario **B**, quien a su vez lo concede al usuario **C**. ¿Que ocurriría si ahora **A** revoca **p** de **B**?. Si el privilegio no fue concedido con la opción CASCADE, la correspondiente sentencia REVOKE no ejecutaría con éxito. Si el privilegio sí fue concedido con la opción CASCADE, la REVOKE se ejecutaría con éxito y además se revocaría el privilegio al usuario **C**.

Si se elimina un objeto, se revocan automáticamente todos los privilegios concedidos sobre el objeto.

Ejemplos de concesión y eliminación de privilegios de objeto

Conceder al usuario *usu1* el privilegio de actualizar la tabla *emp*

```
GRANT UPDATE ON emp TO usu1;
```

Conceder al usuario *usu2* el derecho de modificar solamente la columna *sal* de la tabla *emp*

GRANT UPDATE (*sal*) ON *emp* TO *usu2*;

Conceder todos los derechos a *usu3* sobre la tabla *emp*

GRANT ALL ON *emp* TO *usu3*;

Quitar a *usu1* el derecho de actualizar la tabla *emp*

REVOKE UPDATE ON *emp* FROM *usu1*;

Quitar a *usu3* los derechos de consulta y actualización sobre la tabla *emp*

REVOKE SELECT, UPDATE ON *emp* FROM *usu3*;

Un usuario puede conceder cualquier privilegio de objeto sobre cualquier objeto de su propiedad a cualquier usuario o rol. Si la concesión incluye la cláusula WITH GRANT OPTION, el usuario que recibe el privilegio puede a su vez concedérselo a otros usuarios. Si no incluye la cláusula, dicho usuario sólo puede usar el privilegio, pero no concedérselo a otros.

- **Privilegios del sistema**

Un **privilegio del sistema** es el derecho para realizar una acción concreta en un tipo concreto de objeto. Por ejemplo, los privilegios para crear tablespaces (CREATE TABLESPACE) o para eliminar filas en cualquier tabla de la base de datos (DELETE ANY TABLE), son ejemplos de privilegios del sistema. Hay más de 60 privilegios del sistema.

Los privilegios del sistema son concedidos o revocados a usuarios y roles utilizando las sentencias GRANT y REVOKE.

Sintaxis de la sentencia GRANT

**GRANT {privilegios del sistema |roles }
TO {lista de usuarios | roles | PUBLIC} [WITH ADMIN OPTION]**

Solo los usuarios que tienen concedido el privilegio GRANT ANY PRIVILEGE pueden conceder o revocar privilegios del sistema a otros usuarios. Cuando un usuario tiene concedido un privilegio del sistema con la opción WITH ADMIN OPTION, puede conceder o revocar este privilegio a otros usuarios.

Sintaxis de la sentencia REVOKE

REVOKE {privilegios del sistema | roles} FROM {lista de usuarios | roles | PUBLIC}

Ejemplos de concesión y eliminación de privilegios del sistema

Con la siguiente sentencia se permite al usuario *profesor* crear sinónimos en cualquier esquema. También, puede conceder este privilegio a cualquier otro usuario.

GRANT CREATE ANY SYNONYM TO *profesor* WITH ADMIN OPTION;

El usuario *usu1* puede eliminar tablas en cualquier esquema

GRANT DROP ANY TABLE TO *usu1*;

Quitar el derecho de eliminar tablas a los usuarios *usu1* y *usu2* en todos los esquemas que no sean los suyos, dicho de otra forma, para que los usuarios *usu1* y *usu2* no puedan eliminar tablas que no les pertenezcan, se utiliza la siguiente sentencia REVOKE

REVOKE DROP ANY TABLE FROM *usu1, usu2*;

- ***Funcionalidad de los Roles***

- A un rol se le pueden conceder privilegios del sistema o de objeto.
- A un rol se le pueden conceder otros roles. No obstante, un rol no puede concederse a si mismo ni de forma circular (por ejemplo, el rol A no se puede conceder al rol B si el rol B ha sido previamente concedido al rol A).
- Cualquier rol puede ser concedido a cualquier usuario.

- Cada rol concedido a un usuario puede, en un momento dado, estar habilitado o deshabilitado. Oracle permite a los usuarios habilitar y deshabilitar roles para lograr distintas disponibilidades de privilegios.
- Un rol concedido indirectamente (un rol concedido a otro rol) puede ser explícitamente habilitado o deshabilitado por el usuario. Sin embargo, al habilitar un rol que contiene otros roles, implícitamente se habilitan todos los roles concedidos indirectamente por el rol directamente concedido.
- Cualquier usuario con el privilegio del sistema GRANT ANY ROLE puede conceder o revocar cualquier rol a otros usuarios o roles. Para conceder o revocar roles se usan las sentencias GRANT y REVOKE.
- La sentencia CREATE ROL permite crear un rol con un cierto nombre. El rol se crea vacío, los privilegios se le asignan posteriormente con la sentencia GRANT.
- La sentencia SET ROL permite habilitar y deshabilitar roles.
- A través de la sentencia ALTER USER se pueden establecer los roles por defecto, o sea, aquellos roles concedidos al usuario que se habilitan automáticamente al iniciar la sesión.
- Al crear una base de datos Oracle, se definen automáticamente los roles CONNECT, RESOURCE, DBA, EXP_FULL_DATABASE y IMP_FULL_DATABASE. Estos roles existen por compatibilidad con versiones anteriores de Oracle y se pueden manejar igual que cualquier otro rol.

5.2. Autenticación de usuarios.

Cada base de datos ORACLE tiene una lista de usuarios válidos. Para acceder a la base de datos, un usuario debe de ejecutar una aplicación (SQL*Plus, FORMS, etc.) y conectarse a una instancia de la base de datos usando un nombre de usuario y una contraseña validos.

La validación del nombre y la contraseña dada en el proceso de conexión, lo que se conoce como autenticación del usuario, puede ser realizada por el S.G.B.D. o por el sistema operativo.

En el caso de autenticación por el sistema operativo el S.G.B.D. se limita a comprobar que el identificativo del usuario que intenta acceder a la base de datos está definido en esa base. El sistema operativo se había ocupado, en su momento, de verificar el nombre y la contraseña del usuario.

Las sentencias CREATE USER y ALTER USER permiten definir el método de autenticación que utilizará cada usuario.

Por simplicidad, normalmente se usará uno de los dos métodos para todos los usuarios, aunque es posible usar métodos distintos según los usuarios.

Los nombres de usuario, y en su caso las contraseñas, son creados y modificados con las sentencias CREATE USER, ALTER USER y DROP USER.

6. Descripción de las tablas utilizadas en estas Unidades.

```
CREATE TABLE DEPT
(DEPTNO NUMBER(2) CONSTRAINT PK_DEPT PRIMARY KEY,
DNAME VARCHAR2(14) ,
LOC VARCHAR2(13) ) ;

CREATE TABLE EMP
(EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
ENAME VARCHAR2(10),
JOB VARCHAR2(9),
MGR NUMBER(4),
HIREDATE DATE,
SAL NUMBER(7,2),
COMM NUMBER(7,2),
DEPTNO NUMBER(2) CONSTRAINT FK_DEPTNO REFERENCES DEPT);

INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK');
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
INSERT INTO DEPT VALUES (30,'SALES','CHICAGO');
INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON');
INSERT INTO EMP VALUES
(7369,'SMITH','CLERK',7902,to_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);
INSERT INTO EMP VALUES
(7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981','dd-mm-yyyy'),1600,300,30);
INSERT INTO EMP VALUES
(7521,'WARD','SALESMAN',7698,to_date('22-2-1981','dd-mm-yyyy'),1250,500,30);
INSERT INTO EMP VALUES
(7566,'JONES','MANAGER',7839,to_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);
INSERT INTO EMP VALUES
(7654,'MARTIN','SALESMAN',7698,to_date('28-9-1981','dd-mm-yyyy'),1250,1400,30);
INSERT INTO EMP VALUES
(7698,'BLAKE','MANAGER',7839,to_date('1-5-1981','dd-mm-yyyy'),2850,NULL,30);
INSERT INTO EMP VALUES
(7782,'CLARK','MANAGER',7839,to_date('9-6-1981','dd-mm-yyyy'),2450,NULL,10);
INSERT INTO EMP VALUES
(7788,'SCOTT','ANALYST',7566,to_date('13-JUL-87')-85,3000,NULL,20);
```

```
INSERT INTO EMP VALUES
(7839, 'KING', 'PRESIDENT', NULL, to_date('17-11-1981', 'dd-mm-yyyy'), 5000, NULL, 10);
INSERT INTO EMP VALUES
(7844, 'TURNER', 'SALESMAN', 7698, to_date('8-9-1981', 'dd-mm-yyyy'), 1500, 0, 30);
INSERT INTO EMP VALUES
(7876, 'ADAMS', 'CLERK', 7788, to_date('13-JUL-87')-51, 1100, NULL, 20);
INSERT INTO EMP VALUES
(7900, 'JAMES', 'CLERK', 7698, to_date('3-12-1981', 'dd-mm-yyyy'), 950, NULL, 30);
INSERT INTO EMP VALUES
(7902, 'FORD', 'ANALYST', 7566, to_date('3-12-1981', 'dd-mm-yyyy'), 3000, NULL, 20);
INSERT INTO EMP VALUES
(7934, 'MILLER', 'CLERK', 7782, to_date('23-1-1982', 'dd-mm-yyyy'), 1300, NULL, 10);
COMMIT;
```

Desarrollo de Aplicaciones Informáticas

materiales didácticos de aula



UNIÓN EUROPEA
Fondo Social Europeo



Gobierno del Principado de Asturias
Consejería de Educación y Ciencia



FORMACIÓN PROFESIONAL
Principado de Asturias