U.T. IX.- CREACIÓN DE OBJETOS EN SQL:

VISTAS, ÍNDICES Y PROCEDIMIENTOS.

1.- VISTAS

- Una vista es una tabla temporal cuyo contenido está definido por una consulta.
  - ✓ El script (bloque de código) creado por la consulta se dota de un nombre y se almacena en la BD.
  - ✓ Una vista puede utilizarse en una instrucción SQL de modo similar a como se utiliza una tabla.
  - ✓ Cuando, durante la ejecución de una instrucción SQL, el sistema encuentra el nombre de una vista, ejecuta el script que tiene almacenado con dicho nombre y crea una tabla en memoria interna con el resultado de dicha ejecución.

1.- VISTAS

- Dentro de una vista puede hacerse referencia a una o varias tablas, o bien a otras vistas.

- Las vistas suelen utilizarse para centrar, simplificar y personalizar la percepción de la BD para cada usuario.

- La utilización de vistas permite que los distintos usuarios y/o programas de aplicación tengan acceso únicamente a los datos contenidos en ellas, pero no al resto de datos contenidos en las tablas base subyacentes.

1.1.- CREACIÓN DE VISTAS.

- **Sintaxis:**

CREATE VIEW [< base_de_datos > .] [< propietario > .]
nombre_vista [({ nombre_columna } [, …])]
AS <sentencia SELECT>

- **Nombre_vista:** ha de ser único en la BD.
- **Nombre_columna:** es el identificador de una columna en la vista.
  - Si no se especifica, las columnas de la vista adquieren el nombre especificado en la lista de selección de **SELECT**.
- **AS:** Indica las acciones que se van a llevar a cabo cuando se ejecute la vista. Estas acciones tendrán la forma de una sentencia **SELECT**, pudiendo ser ésta de cualquier complejidad, aunque con ciertas restricciones.
  - Por ejemplo, no puede utilizarse la cláusula **ORDER BY** salvo que se utilice la cláusula **TOP n** en la lista de selección.

1.1.- CREACIÓN DE VISTAS.

- Cuando se crea una vista, el sistema realiza las siguientes acciones:
  - ✓ Almacena su nombre en la tabla **sysobjects**.
  - ✓ Almacena la información acerca de sus columnas en la tabla **syscolumns**.
  - ✓ Almacena el texto de la instrucción **CREATE VIEW** en la tabla **syscomments**.

1.1.- CREACIÓN DE VISTAS.

- Cuando se hace referencia a una vista dentro de una instrucción SQL, el sistema comprueba que todos los objetos a los que se hace referencia en **CREATE VIEW** existen, que son válidos en el contexto de la instrucción y, en caso de que la consulta implique modificación de datos, que dicha modificación no infringe ninguna regla de integridad.

  - ✓ Las comprobaciones correctas convierten la acción de la instrucción SQL en una acción contra las tablas subyacentes, las no correctas devuelven un mensaje de error.

1.1.- CREACIÓN DE VISTAS.

- Si queremos modificar datos a través de una vista, a la hora de crear ésta hay que tener en cuenta lo siguiente:
  - ✓ No se debe especificar **DISTINCT**.
  - ✓ Aunque la cláusula **FROM** puede contener varias tablas, en cada instrucción **INSERT** o **UPDATE** sólo podrá hacerse referencia a los campos de una de ellas.
  - ✓ La lista de selección sólo puede contener nombres de columnas.
  - ✓ La cláusula **WHERE** no debe incluir una subconsulta.
  - ✓ No puede incluir una cláusula **GROUP BY**.

1.2.- MODIFICACIÓN DE VISTAS.

- **Sintaxis:**

ALTER VIEW [< base_de_datos > .] [< propietario > .]
nombre_vista [({ nombre_columna } [,…])]
AS <sentencia SELECT>

- ✓ Esta sentencia permite modificar una vista creada previamente.

1.3.- BORRADO DE VISTAS.

- **Sintaxis:**

DROP VIEW { nombre_vista } [,…]

- ✓ Esta sentencia permite borrar una o más vistas.

2.- ÍNDICES.

- Un fichero índice es una estructura asociada a una tabla o vista que acelera la búsqueda de datos.
- Cada registro de un índice está compuesto por dos campos:
  - ✓ Una **clave de búsqueda:** Contiene el valor de una o varias columnas por cada fila de una tabla.
  - ✓ Un **puntero:** Contiene la dirección de la fila en el dispositivo de almacenamiento.
- La ventaja de crear índices es que las búsquedas indexadas son más rápidas que las secuenciales.
- La desventaja es que, además de ocupar espacio de almacenamiento, necesitan estar permanentemente actualizados, lo que ralentiza los procesos de modificación de los datos.

2.1.- CREACIÓN DE ÍNDICES.

- **Sintaxis:**

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX nombre_índice
ON nombre_tabla ({columna [ASC | DESC] } [, …])

- ✓ **UNIQUE:** Crea un índice único, es decir, no pueden existir dos registros con el mismo valor de clave.
- ✓ **CLUSTERED:** Crea un índice donde el orden físico de sus registros coincide con el orden físico de las filas de la tabla indexada.
  - Una tabla sólo puede contener un índice agrupado.
  - Por defecto, al crear una tabla, el sistema crea un índice **clustered** con el o los campos que forman la clave primaria.
- ✓ **NONCLUSTERED:** Crea un índice donde el orden físico de la filas de la tabla indexada no coincide con el orden físico de los registros del índice.
  - Pueden crearse cualquier número de índices no agrupados para una misma tabla. Es la opción por defecto.

2.2.- BORRADO DE ÍNDICES.

- **Sintaxis:**

DROP INDEX nombre_tabla.nombre_índice [,…]

- ✓ Esta sentencia permite borrar uno o más índices.

3.- PROCEDIMIENTOS.

- Un procedimiento en SQL es un bloque de instrucciones dotado de un nombre que se almacena en la BD.
  - ✓ Se utilizan para encapsular tareas repetitivas.
- Son similares a los procedimientos de otros lenguajes de programación, ya que pueden contener:
  - ✓ Instrucciones que realizan operaciones en la BD.
  - ✓ Llamadas a otros procedimientos.
  - ✓ Variables declaradas por el usuario.
  - ✓ Ejecución condicional: alternativa y repetitiva.
  - ✓ Parámetros de entrada.
  - ✓ Parámetros de salida.

3.- PROCEDIMIENTOS.

- Cuando se crea un procedimiento, el sistema analiza las instrucciones que contiene desde un punto de vista sintáctico. Si no detecta errores, almacena su nombre en **sysobjects** y su texto en **syscomments.**

- Cuando se ejecuta por primera vez, un procedimiento pasa por una serie de fases (optimización y compilación) cuyo resultado es la creación de un plan de ejecución optimizado que el sistema guarda en la caché de procedimientos.

- Las siguientes veces que se ejecute el procedimiento el sistema utiliza directamente el plan de ejecución creado.

3.1.- OPERACIONES CON PROCEDIMIENTOS.

- **Creación:**

CREATE PROC [EDURE] NombreProcedimiento

[<Declaración_de_Parámetros>]

AS {<instrucción SQL>} …

- ✓ Un procedimiento puede contener cualquier número y tipo de instrucciones SQL excepto algunas de creación de determinados objetos, entre ellos las vistas.

3.1.- OPERACIONES CON PROCEDIMIENTOS.

- **Ejecución:**

EXEC [UTE] NombreProcedimiento

- **Modificación:**

ALTER PROC [EDURE] NombreProcedimiento

[<Declaración_de_Parámetros>]

AS {<instrucción SQL>} …

- **Borrado:**

DROP PROC [EDURE] NombreProcedimiento

4.- VARIABLES INDEPENDIENTES.

- En SQL pueden declararse variables que no formen parte de ninguna tabla.

- Declaración de variables:

DECLARE {@nombre_variable TipoDatos} [,…]

- Asignación de valores:

SET @nombre_variable = expresión.

- Visualización del contenido de una variable:

SELECT {@nombre_variable [AS identificador]} [,…]

5.- PROCEDIMIENTOS CON PARÁMETROS DE ENTRADA.

- Los parámetros de entrada permiten pasar información desde el exterior al procedimiento.

- Para definir un procedimiento con parámetros de entrada hay que declarar éstos como variables independientes dentro de **<Declaración_de_Parámetros>**

  - ✓ Pueden asignárseles valores iniciales (constantes ó NULL).

- Los parámetros son locales al procedimiento.

  - ✓ Sólo existen durante el tiempo de ejecución de éste.

- La información acerca de los parámetros se almacena en la tabla **syscolumns**.

5.1.- EJECUCIÓN DE PROCEDIMIENTOS CON PASO DE VALORES.

- **Paso de valores por nombre de parámetro:**

EXEC NombreProcedimiento

{@NombreParámetro = Valor} [, …]

- ✓ En el paso por nombre de parámetro los valores se pueden especificar en cualquier orden y pueden omitirse los que acepten valores nulos o tengan un valor inicial.

5.1.- EJECUCIÓN DE PROCEDIMIENTOS CON PASO DE VALORES.

- **Paso de valores por posición de parámetro:**

EXEC NombreProcedimiento Valor1, Valor2, …

- En el paso de valores por posición de parámetro es necesario especificar los valores en el mismo orden en el que se declararon los parámetros.
  - ✓ Pueden omitirse los valores para los parámetros con un valor inicial o que acepten valores nulos, siempre que hayan sido declarados al final de la lista de declaraciones.
  - ✓ Si alguno de estos parámetros fue declarado en medio de la lista, habrá que especificar DEFAULT o NULL, respectivamente.

6.- PROCEDIMIENTOS CON PARÁMETROS DE SALIDA.

- Los parámetros de salida se utilizan para almacenar el/los resultados de la ejecución de un procedimiento.
  - ✓ Son globales al procedimiento, es decir, siguen existiendo una vez terminada su ejecución.
- Un parámetro de salida se declara dentro de **<Declaración_de_Parámetros>,** antes de la declaración de los parámetros de entrada y seguido de la palabra reservada **OUTPUT.**
- Cuando se ejecute el procedimiento, debe declararse previamente un parámetro de salida *actual* que tomará el valor del parámetro de salida *formal* declarado dentro del procedimiento.