

Tema 3: El lenguaje SQL (II)

Asignatura: Bases de datos

CS Desarrollo de Aplicaciones Web



Introducción

- En este capítulo veremos aspectos como:
 - Uso de funciones predefinidas
 - Inserción de datos
 - Actualización de datos
 - Borrado de datos

USO DE FUNCIONES PREDEFINIDAS EN CONSULTAS SQL

Uso de funciones predefinidas en consultas SQL

Funciones integradas

- Las funciones integradas son aquellas que devuelven información facilitada por el RDBMS: la fecha y hora actual, el nombre de usuario,...

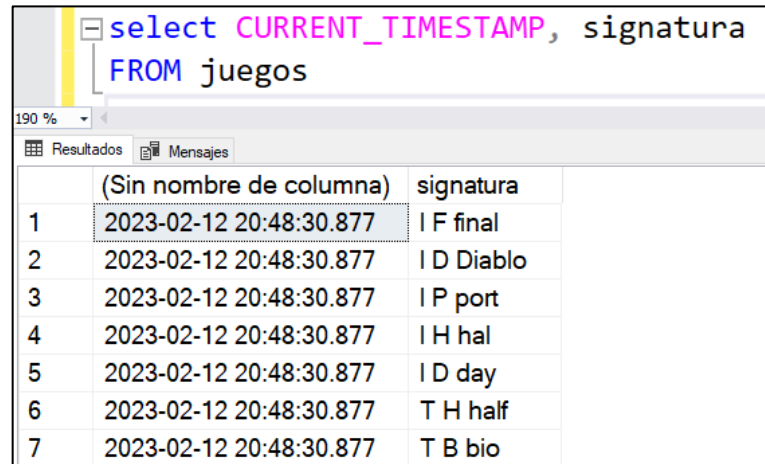
Nombre	Descripción
CURRENT_TIMESTAMP	Devuelve la fecha y hora actuales
CURRENT_USER Y USER	Identifica al usuario que está operando sobre la base de datos
SESSION_USER	Facilita el identificador del usuario si este difiere de CURRENT_USER
SYSTEM_USER	Identifica al usuario según la cuenta del sistema operativo

Uso de funciones predefinidas en consultas SQL

- Consideraciones previas:
 - **Todas** las funciones anteriores **forman parte del estándar SQL**, pero **no todos los RDBMS las contemplan** o bien les dan otros nombres.
 - Hay que tener en cuenta si el usuario usa una cuenta para iniciar sesión en el sistema operativo y es éste el que se encarga de autenticarlo ante el RDBMS o, por el contrario, el usuario tiene una cuenta específica para acceder a la base de datos.
- Estas funciones **devuelven los datos como si se tratase de columnas**, aunque en realidad no están almacenados en ninguna tabla.

Uso de funciones predefinidas en consultas SQL

- Un ejemplo de uso sería el siguiente, donde se combina una de estas funciones con una consulta corriente. En este caso se ha utilizado la función **CURRENT_TIMESTAMP**, que devuelve la fecha y la hora sin ninguna información adicional:



```
select CURRENT_TIMESTAMP, signatura
FROM juegos
```

190 %

Resultados Mensajes

	(Sin nombre de columna)	signatura
1	2023-02-12 20:48:30.877	I F final
2	2023-02-12 20:48:30.877	I D Diablo
3	2023-02-12 20:48:30.877	I P port
4	2023-02-12 20:48:30.877	I H hal
5	2023-02-12 20:48:30.877	I D day
6	2023-02-12 20:48:30.877	T H half
7	2023-02-12 20:48:30.877	T B bio

Uso de funciones predefinidas en consultas SQL

Funciones de cadena

- Las funciones de cadena son todas aquellas que **permiten operar sobre secuencias o cadenas de caracteres**. Ya conocemos una de ellas, `SUBSTRING`, pero además de ésta el estándar SQL cuenta con otras entre las que destacan:
 - **`CHAR_LENGTH/CHARACTER_LENGTH`**: Proporciona el número de caracteres existentes en el dato facilitado como argumento, lo que se conoce como **longitud de la cadena**.
 - **`CONVERT`**: Convierte la **cadena original a una representación distinta** usando el mismo conjunto de caracteres
 - **`LOWER`**: **Convierte a minúsculas** los caracteres que estén en mayúsculas

Uso de funciones predefinidas en consultas SQL

- **POSITION:** Facilita la posición en la que aparece una cierta secuencia de caracteres dentro de otra cadena
- **SUBSTRING:** Extrae parte de una cadena
- **TRANSLATE:** Efectúa la conversión de cadenas entre diferentes conjuntos de caracteres
- **TRIM:** Elimina espacios sobrantes en la cadena
- **UPPER:** Convierte a mayúsculas los caracteres que estén en minúsculas

Uso de funciones predefinidas en consultas SQL

Longitud de la cadena

- A la hora de crear una tabla de una base de datos **se tienen que especificar el tipo de datos** que va a permitir almacenar. Entre ellos se encuentran las secuencias de caracteres para los cuales suele haber habitualmente dos tipos:
 - CHAR: Asocia una **longitud fija** al dato que va a almacenar
 - VARCHAR: Establece una **longitud máxima** para el almacenamiento, **pero utiliza sólo el espacio necesario** para la cadena que vaya a contener. **La longitud de ésta se puede obtener** mediante la función `CHAR_LENGTH` o su equivalente `CHARACTER_LENGTH`.

Uso de funciones predefinidas en consultas SQL

- Para los RDBMS la función será será:
 - MariaDB: Reconoce tanto `CHAR_LENGTH` como `CHARACTER_LENGTH`
 - Oracle: Reconoce `LENGTH`
 - SQL Server: Utiliza `LEN`
 - Access: Al igual que SQL Server también utiliza `LEN`
- En cualquiera de los casos, la función se comporta de la misma manera devolviendo como resultado un número entero con la longitud.

Uso de funciones predefinidas en consultas SQL

- Para poder ver su uso se plantea el siguiente ejemplo.
- Se trata de averiguar la longitud del título de cada juego existente en la gamoteca

```
SELECT signatura, LEN(titulo) AS Longitud  
FROM juegos
```

190 %

Resultados Mensajes

	signatura	Longitud
1	I F final	17
2	I D Diablo	9
3	I P port	8
4	I H hal	9
5	I D day	19
6	T H half	11
7	T B bio	8

Uso de funciones predefinidas en consultas SQL

Eliminación de caracteres sobrantes

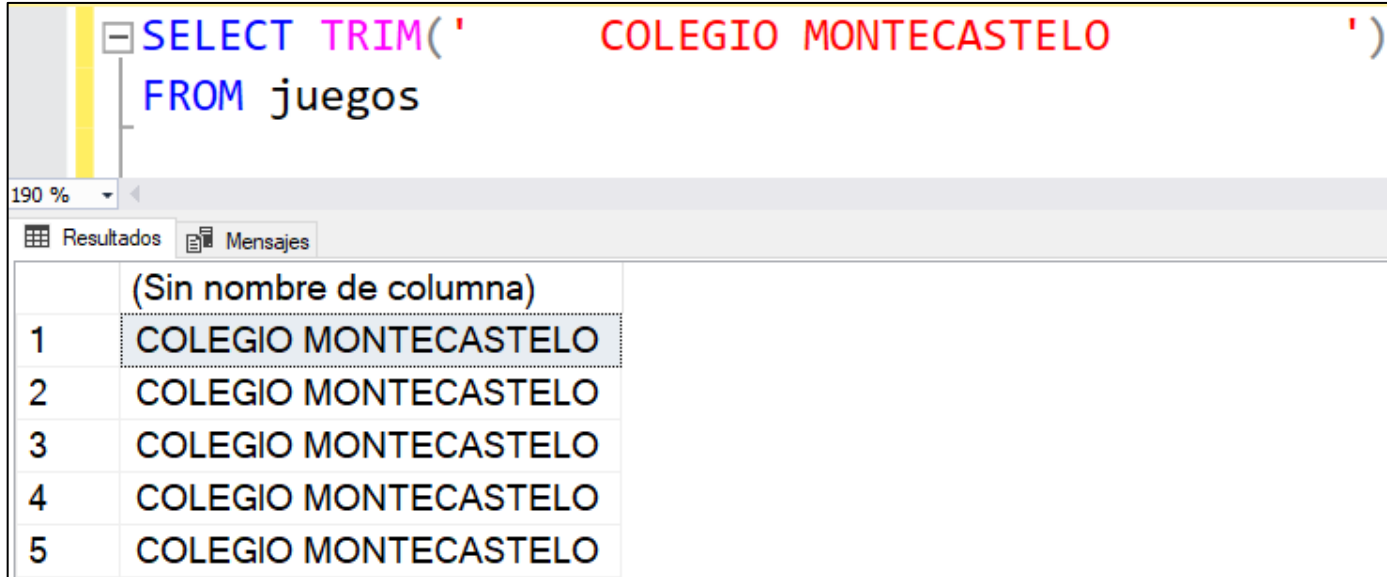
- Suele suceder que **a la hora de introducir datos** en una base de datos **se incorporen, de manera involuntaria** o incorrecta, **caracteres en blanco sobrantes al inicio o al final**. Estos caracteres pueden ser eliminados gracias a la función `TRIM`.
- La sintaxis es la siguiente:

```
TRIM([characters FROM ]string)
```

Uso de funciones predefinidas en consultas SQL

- Teniendo en cuenta que los corchetes indican que su contenido es opcional, la forma más sencilla de utilizar esta función sería: `TRIM(valor)`. En este caso se eliminarían los espacios en blanco que existen al inicio y al final del valor.
- SQL Server y Access cuentan con las funciones **LTRIM** y **RTRIM** que eliminan espacios iniciales y finales respectivamente sin aceptar mas argumentos que el valor sobre el que se va a operar.

Uso de funciones predefinidas en consultas SQL



The screenshot shows a SQL query editor with the following text:

```
SELECT TRIM('          COLEGIO MONTECASTELO          ')  
FROM juegos
```

Below the editor, the 'Resultados' (Results) tab is active, displaying a table with 5 rows. The first row is highlighted. The table has one column labeled '(Sin nombre de columna)'.

	(Sin nombre de columna)
1	COLEGIO MONTECASTELO
2	COLEGIO MONTECASTELO
3	COLEGIO MONTECASTELO
4	COLEGIO MONTECASTELO
5	COLEGIO MONTECASTELO

Uso de funciones predefinidas en consultas SQL

Conversión de caracteres

- Las funciones `UPPER` y `LOWER` se utilizan para conversión a mayúsculas y minúsculas respectivamente de todos los caracteres alfabéticos que existan en la cadena que se les pasa como argumento.
- Dependiendo del RDBMS es posible que **determinados caracteres**, como las vocales acentuadas y eñes, **no sean tratados adecuadamente** a la hora de realizar la conversión.
- En cuanto a las funciones `CONVERT` y `TRANSLATE`, cuya finalidad es convertir cadenas entre diferentes conjuntos de caracteres, ninguno de los RDBMS las implementa de acuerdo al estándar. Será necesario consultar la documentación en cada caso.

Uso de funciones predefinidas en consultas SQL

- Como ejemplo vamos a obtener la signature de todos los juegos en minúsculas y el título en mayúsculas.

```
SELECT LOWER(signature), UPPER(titulo)
FROM juegos
```

190 %

Resultados Mensajes

	(Sin nombre de columna)	(Sin nombre de columna)
1	i f final	FINAL FANTASY VII
2	i d diablo	DIABLO II
3	i p port	PORTAL 2
4	i h hal	HALF-LIFE
5	i d day	DAY OF THE TENTACLE
6	t h half	HALF-LIFE 2
7	t b bio	BIOSHOCK
8	t t the	THE WITCHER

Uso de funciones predefinidas en consultas SQL

Posición de una cadena en otra

- Para poder averiguar la posición de una cadena dentro de otra cadena utilizamos la función CHARINDEX, cuya sintaxis es:

```
CHARINDEX(substring, string, start)
```

- Si la subcadena no aparece dentro de la cadena, la función devuelve el valor 0.

Uso de funciones predefinidas en consultas SQL

- Vamos a buscar en nuestra gamoteca todos los juegos cuyo título contiene las letras “de”, indicando en que posición se encuentra dicha subcadena.

```
SELECT signatura, titulo, CHARINDEX('de',titulo) AS Posición
FROM juegos
WHERE CHARINDEX('de',titulo) > 0
```

190 %

Resultados Mensajes

	signatura	titulo	Posición
1	G T walk	The Walking Dead	13
2	G T elder	The Elder Scrolls	7
3	G T destiny	Destiny	1

Uso de funciones predefinidas en consultas SQL

Funciones de fechas

- Ya hemos visto la función informativa `CURRENT_TIMESTAMP`.
- Además, SQL proporciona la función `DATEPART`. La utilidad de esta función es recuperar una parte de la fecha u hora según la siguiente sintaxis:

```
DATEPART(interval, date)
```

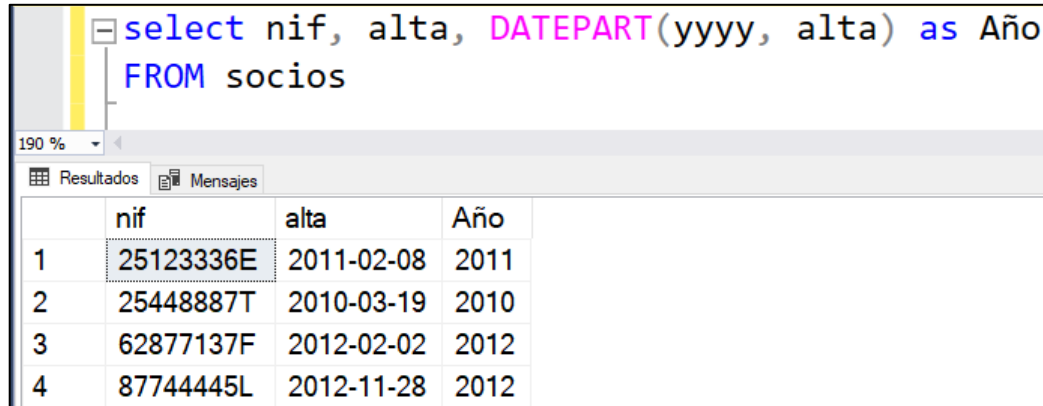
- Donde `interval` puede ser cualquiera de los identificadores de la siguiente tabla, `date` puede ser cualquier valor devuelto por `CURRENT_TIMESTAMP` o una fecha u hora que tengamos almacenada en la tabla.

Uso de funciones predefinidas en consultas SQL

Identificador	Valor devuelto	Equivalente estándar
d	Día del mes	DAY
m	Número del mes	MONTH
yyyy	Año	YEAR
hh	Horas	HOURL
n	Minutos	MINUTE
s	Segundos	SECOND

Uso de funciones predefinidas en consultas SQL

- Teniendo en cuenta estas funciones vamos a obtener en una columna el año de alta de cada socio de nuestra `gameteca`:



```
select nif, alta, DATEPART(yyyy, alta) as Año
FROM socios
```

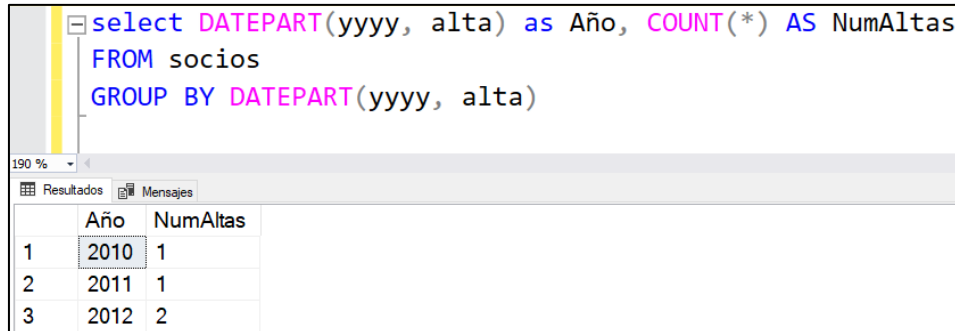
190 %

Resultados Mensajes

	nif	alta	Año
1	25123336E	2011-02-08	2011
2	25448887T	2010-03-19	2010
3	62877137F	2012-02-02	2012
4	87744445L	2012-11-28	2012

Uso de funciones predefinidas en consultas SQL

- Como el resto de las funciones de SQL, DATEPART puede ser utilizada para filtrar las filas obtenidas, por ejemplo, para recuperar los socios de un cierto año o mes; ordenarlos, agruparlas...
- En la siguiente consulta mostramos como obtener el número de altas registradas cada año.



```
select DATEPART(yyyy, alta) as Año, COUNT(*) AS NumAltas
FROM socios
GROUP BY DATEPART(yyyy, alta)
```

190 %

Resultados Mensajes

	Año	NumAltas
1	2010	1
2	2011	1
3	2012	2

Uso de funciones predefinidas en consultas SQL

Funciones numéricas: Redondeo y valor absoluto

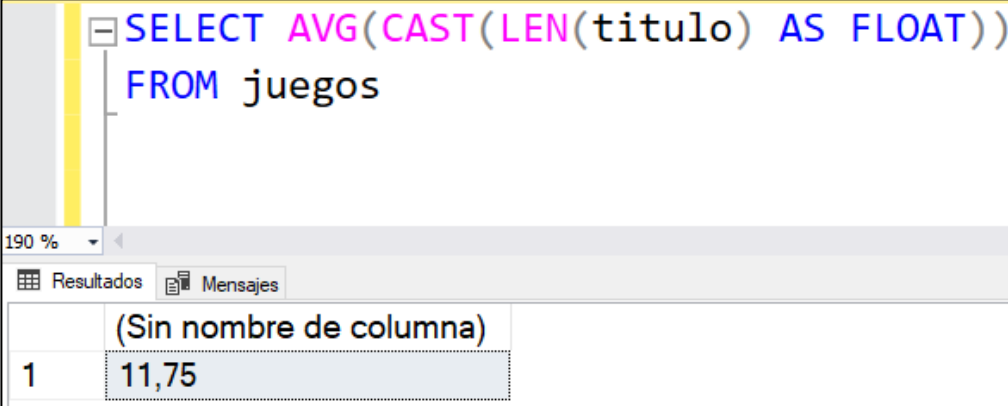
- Las funciones de redondeo sirven para operar sobre números que tienen una parte decimal que deseamos eliminar, quedándonos con el número entero más próximo inmediatamente inferior o superior, según los casos.
 - Función **FLOOR**: Devuelve el número entero inmediatamente inferior al número entregado como argumento
 - Función **CEIL** o **CEILING**: Devuelve el número entero inmediatamente superior al número entregado como argumento

Uso de funciones predefinidas en consultas SQL

- Con respecto a estas funciones el comportamiento de los distintos RDBMS es el siguiente:
 - MaríaDB interpreta perfectamente las funciones `FLOOR`, `CEIL` y `CEILING`
 - Oracle no reconoce `CEILING` pero si `CEIL`
 - SQL Server no reconoce `CEIL` pero si `CEILING`
 - Access no cuenta con estas funciones. En su lugar utiliza la función `ROUND` para los redondeos

Uso de funciones predefinidas en consultas SQL

- En el siguiente ejemplo vamos a ver la diferencia entre FLOOR y CEIL. Para ello vamos a calcular la media de la longitud del nombre de autor de cada juego.



```
SELECT AVG(CAST(LEN(titulo) AS FLOAT))  
FROM juegos
```

	(Sin nombre de columna)
1	11,75

Uso de funciones predefinidas en consultas SQL

- La función FLOOR prescindirá de la parte decimal y se quedará con el 11, mientras que la función CEIL tomará el número inmediatamente superior, que es el 12

```
SELECT FLOOR(AVG(CAST(LEN(titulo) AS FLOAT)))  
FROM juegos
```

(Sin nombre de columna)	
1	11

```
SELECT CEILING(AVG(CAST(LEN(titulo) AS FLOAT)))  
FROM juegos
```

(Sin nombre de columna)	
1	12

Uso de funciones predefinidas en consultas SQL

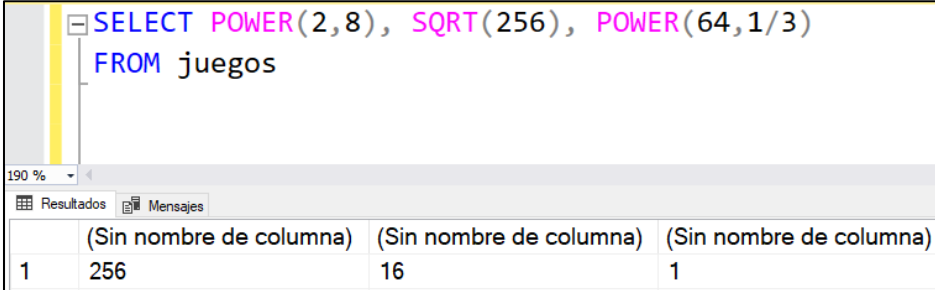
- El **comportamiento** anterior **se mantiene** al operar **con números negativos**.
- El **valor absoluto** de un número es su valor sin signo alguno. Se puede calcular el valor absoluto de cualquier número **mediante la función** `ABS`. Esta función está reconocida por los cuatro RDBMS a los que nos referimos continuamente.

Raíces y potencias

- Para realizar estas operaciones matemáticas SQL incorpora dos funciones:
 - **POWER**: Esta función toma una base y un exponente, devolviendo el resultado de la correspondiente operación de potenciación
 - **SQRT**: Devuelve como resultado la raíz cuadrada del número que se le pasa como argumento

Uso de funciones predefinidas en consultas SQL

- El resultado de algunas operaciones aritméticas puede diferir según el RDBMS que se utilice, ya que **no todos operan con la misma precisión por defecto**.
- En el siguiente ejemplo se obtiene el resultado de elevar 2 al exponente 8, obtener la raíz cuadrada de 256 y la raíz cúbica de 64:



```
SELECT POWER(2,8), SQRT(256), POWER(64,1/3)
FROM juegos
```

190 %

Resultados Mensajes

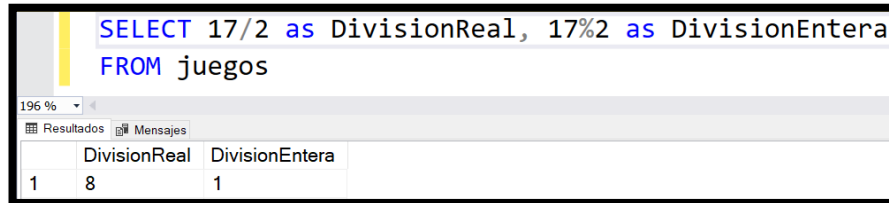
	(Sin nombre de columna)	(Sin nombre de columna)	(Sin nombre de columna)
1	256	16	1

Uso de funciones predefinidas en consultas SQL

Otras funciones matemáticas

- **Operador %:** Obtiene el resto de una división entera al pasarle como argumentos el dividendo y el divisor.
- La sintaxis es:

dividendo % divisor



```
SELECT 17/2 as DivisionReal, 17%2 as DivisionEntera
FROM juegos
```

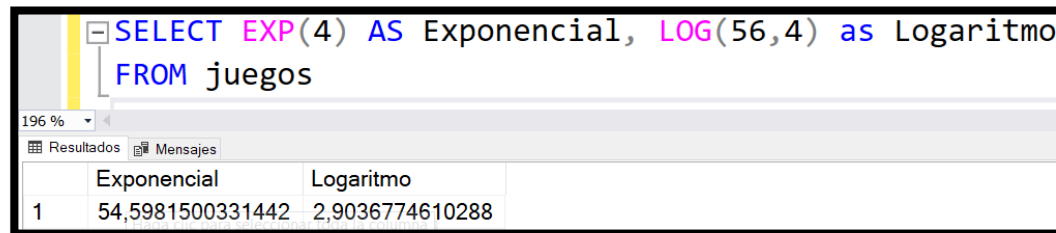
	DivisionReal	DivisionEntera
1	8	1

The screenshot shows a SQL query editor with the query: `SELECT 17/2 as DivisionReal, 17%2 as DivisionEntera FROM juegos`. Below the query, the results are displayed in a table with two columns: `DivisionReal` and `DivisionEntera`. The first row shows the results for the query: `1`, `8`, and `1`.

Uso de funciones predefinidas en consultas SQL

Otras funciones matemáticas

- Función **EXP**: Es similar a POWER pero únicamente necesita como parámetro el exponente, ya que la base es e.
- Función **LOG**: Devuelve el logaritmo natural, es decir, el exponente al que hay que elevar e para obtener el número entregado como argumento.



```
SELECT EXP(4) AS Exponencial, LOG(56,4) as Logaritmo
FROM juegos
```

	Exponencial	Logaritmo
1	54,5981500331442	2,9036774610288

Uso de funciones predefinidas en consultas SQL

Otras funciones SQL

- Función **COALESCE**: Nos permite sustituir los valores nulos que existan en una columna por lo que nos interese. Si tiene un valor, que suele ser el nombre de una columna, contiene `NULL`, entonces se insertará en su lugar el valor sustituto.
- La sintaxis es: `COALESCE(valor, sustituto)`

Uso de funciones predefinidas en consultas SQL

```
SELECT signatura, COALESCE(disponible, 'Indeterminado') AS Disponibilidad
FROM juegos
```

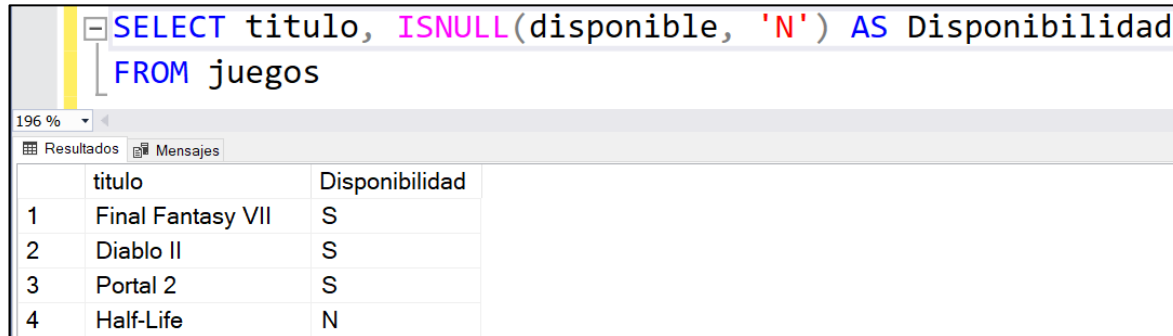
196 %

Resultados Mensajes

	signatura	Disponibilidad
1	I F final	S
2	I D Diablo	S
3	I P port	S
4	I H hal	N
5	I D day	S
6	T H half	S
7	T B bio	N
8	T T the	S
9	G T walk	S
10	G E ense	N
11	G T elder	Sin info

Uso de funciones predefinidas en consultas SQL

- Función **ISNULL**: Compara el contenido de una columna con un cierto valor, y en caso de ser NULL, lo sustituye por ese valor. Supongamos, por ejemplo, que en nuestra gamoteca no aparezcan NULL, sino que salga que no están disponibles.



The screenshot shows a SQL query editor with the following query:

```
SELECT titulo, ISNULL(disponible, 'N') AS Disponibilidad  
FROM juegos
```

Below the query editor, the results window is displayed, showing the output of the query. The results are as follows:

	titulo	Disponibilidad
1	Final Fantasy VII	S
2	Diablo II	S
3	Portal 2	S
4	Half-Life	N

Actividad 1

Crea las queries necesarias para lograr lo siguiente:

1. Obtener la longitud de los nombres de los futbolistas
2. Obtener la fecha actual
3. Convertir el nombre y apellidos de futbolistas a mayúsculas
4. Calcular la suma de goles de un equipo de la tabla Futbolistas
5. Calcular el promedio de goles de un equipo
6. Obtener el número de futbolistas de un equipo
7. Obtener la parte entera de un promedio de goles de cada equipo
8. Obtener la parte decimal de un promedio de goles de cada equipo

DIFICILES

1. Obtener la longitud de SOLO EL NOMBRE de los futbolistas
2. Calcular la suma de goles de un equipo de la tabla Futbolistas y que salgan los equipos con 0 goles
3. Obtener el jugador que menos goles mete y el que más de un equipo (difícil)

20 MINUTOS

INSERCIÓN DE NUEVOS DATOS

Inserción de nuevos datos

La sentencia INSERT

- La cláusula `INSERT` nos permite **agregar filas a una tabla, o en las columnas obtenidas** mediante una **vista** si ésta es actualizable.
- La sintaxis es la siguiente:

```
INSERT INTO tabla[(col1,col2,...)]  
VALUES (val1,val2,...)
```

Inserción de nuevos datos

La sentencia INSERT

- Donde `tabla` es la tabla en la que se van a añadir los datos. Ésta irá **seguida opcionalmente**, entre paréntesis, **de los nombre de las columnas** donde quieren insertarse los valores enumerados tras la palabra clave `VALUES`. Habrá tantos valores como columnas se indiquen, o existan en la tabla, agregándose una nueva fila conteniendo dicho valores.
- Si alguno de ellos incumple una restricción, por ejemplo duplicando un valor que no puede estar repetido o dejando como nula una columna que no puede ser `NULL`, **la operación completa fallará**, obtendremos un mensaje de error y no se producirá ningún cambio en la tabla.

Inserción de nuevos datos

Inserción de valores por posición

- Hemos visto en el apartado anterior que los nombres de las columnas son un parámetro opcional. Si los omitimos la sentencia `INSERT` tomará el siguiente formato:

```
INSERT INTO tabla  
VALUES (val1, val2, ...)
```

- Los valores facilitados tras `VALUES`:
 - Han de aparecer **en el mismo orden en el que se definieron las columnas** de la tabla, existiendo tantos valores como columnas existen en la tabla
 - Deben **coincidir en tipo y no incumplir ninguna de las restricciones** que pudieran haberse establecido

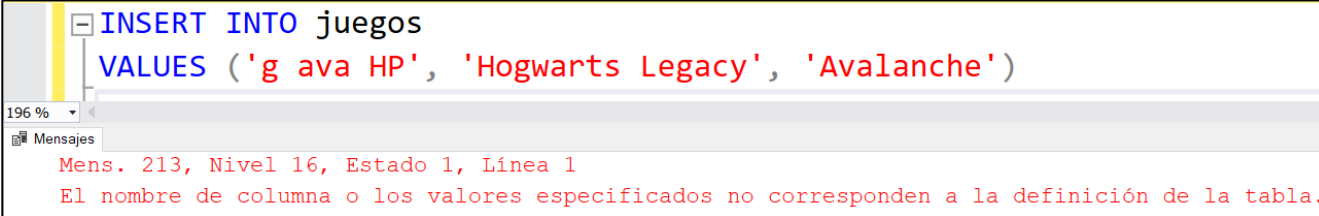
Inserción de nuevos datos

- Como ejemplo vamos a suponer que queremos añadir un nuevo juego a nuestra gamoteca. En este caso deberíamos proporcionar un valor para la columna código, otro para signature, título, creador y disponible, en ese mismo orden.

```
INSERT INTO juegos  
VALUES ('g ava HP', 'Hogwarts Legacy', 'Avalanche', 'S' )
```

Inserción de nuevos datos

- Si entregamos menos valores de los necesarios, el RMDB nos informará del error con un mensaje como el siguiente:



```
INSERT INTO juegos
VALUES ('g ava HP', 'Hogwarts Legacy', 'Avalanche')
```

Mensajes

Mens. 213, Nivel 16, Estado 1, Línea 1

El nombre de columna o los valores especificados no corresponden a la definición de la tabla.

Inserción de nuevos datos

- Algo similar ocurriría al incumplir una restricción, por ejemplo, si intentamos introducir en la columna `código` un valor que ya existe. Esta columna es la clave principal de la tabla y, como tal, no admite valores duplicados:

```
insert into peliculas
values (1, 'Pacific Rin', 120, 2013, 'AA', 'EE', 1)
```

Mensajes

Mens. 2627, Nivel 14, Estado 1, Línea 11
Infracción de la restricción PRIMARY KEY 'PK__pelicula__40F9A207DBA1C97F'.

Inserción de nuevos datos

Inserción de valores por nombre de columna

- En algunos casos puede ser que **necesitemos especificar** tras el nombre de la tabla, entre paréntesis, **los nombres de cada una de las columnas** en las que vamos a insertar datos. Estos **casos** pueden ser:
 - **No conocemos la posición de las columnas** en la tabla para facilitar los valores en el orden correcto
 - **Vamos a aportar menos valores** que columnas existen en la tabla. Esto **no generará un error** por parte del RDBMS **siempre que en dichas columnas no se haya especificado la restricción** NO NULL en la definición.

Inserción de nuevos datos

- Como ejemplo vamos a añadir un nuevo socio a la tabla `socios` de nuestra `gameteca`. Para ello vamos a insertar datos en todas las columnas excepto en las columnas `dirección` y `código postal` para las que no se ha establecido la restricción `NO NULL` tal como se explicó antes.

```
INSERT INTO socios(nif, nombre, apellidos, alta)
VALUES ('12346578Z', 'Michael', 'Schumacher', CURRENT_TIMESTAMP)
```

nsajes

(1 fila afectada)

Inserción de nuevos datos

Obtener la estructura de una tabla

- Es habitual que **tengamos que trabajar sobre una base de datos que no hayamos definido** nosotros, por lo que su estructura nos será desconocida. Para facilitar nuestras consultas tendremos que hacer lo que se denomina una solicitud de información de esquema. Cada RDBMS implementa un mecanismo distinto para esta tarea.
- Los RDBMS almacena la información de estructura en una serie de tablas del sistema que, mediante un sentencia `SELECT`, podemos consultar obteniendo nombre, tipo, longitud y otros atributos de cada columna, así como el nombre de cada tabla, vista, procedimiento almacenado y cualquier otro objeto que pueda existir en la base de datos.

Inserción de nuevos datos

- En el caso de utilizar SQL Server tenemos varias alternativas:
 - Utilizar distintas vistas prefabricadas, como `INFORMATION_SCHEMA.COLUMNS`. En la siguiente sentencia se solicita información sobre la tabla juegos

```
SELECT *  
FROM INFORMATION_SCHEMA.COLUMNS  
WHERE TABLE_NAME='juegos';
```

Inserción de nuevos datos

- En el caso de utilizar SQL Server tenemos varias alternativas:
 - Otra posibilidad es recurrir a los procedimientos almacenados de sistema SQL Server. Uno de ellos `sp_columns`, nos permite recuperar información de las columnas correspondientes a la tabla cuyo nombre asignaremos a `table_name`:

```
sp_columns @table_name='juegos';
```

Inserción de nuevos datos

Valores por defecto y valores nulos

- Como ya hemos visto, en una operación de inserción en la que se especifican los nombres de algunas columnas para insertar datos en ellas, las columnas que no se especifican se rellenan con `NULL`. Esto es así suponiendo que dichas columnas admitan la inclusión de valores nulos.

Inserción de nuevos datos

Valores por defecto y valores nulos

- La inserción de valores nulos en las columnas también puede efectuarse de manera explícita, facilitando la constante `NULL` en la posición que debería ocupar el valor asociado a la columna que corresponda.

```
INSERT INTO socios  
VALUES ('12345678Z', 'Rafael', 'Nadal Parera', NULL, NULL,  
        CURRENT_TIMESTAMP)
```


Inserción de nuevos datos

- También es posible introducir en una columna el valor establecido como contenido por defecto en la definición de la tabla indicando `DEFAULT` en lugar de `NULL`. Si en esa definición no se indicó valor por defecto alguno, el contenido asignado normalmente será `NULL`.

```
INSERT INTO socios  
VALUES ('12345678Z', 'Michael', 'Jordan', DEFAULT, DEFAULT,  
CURRENT_TIMESTAMP)
```

Inserción de nuevos datos

Inserciones y subconsultas

- La combinación mas habitual de subconsultas e inserciones tiene lugar cuando se quieren tomar datos de una tabla para introducirlos en otra. En ese caso se emplea la siguiente sintaxis:

```
INSERT INTO tabla [(columnas)]  
SELECT columnas FROM tabla  
[WHERE filtro]
```

Inserción de nuevos datos

Inserciones y subconsultas

- Un ejemplo podría ser si hubiésemos creado una tabla para alojar la signatura y título de aquellos juegos que están disponibles.
- En ese caso podríamos introducir en ellas los juegos correspondientes mediante la sentencia:

```
INSERT INTO disponibles  
SELECT signatura, titulo  
FROM juegos WHERE disponible='S';
```

Inserción de nuevos datos

Generación automática de códigos

- A la hora de insertar un nuevo juego en nuestra *gameteca*, en el campo código, teníamos que incrementar en una unidad el valor del código del último juego introducido. Esto es debido a que el campo código debe estar constituido de un valor único y que debe ser un número entero y secuencial. Esto supone conocer el último código usado, trabajo que puede llevar a cabo el RDBMS.

Inserción de nuevos datos

- Podríamos hacerlo con la siguiente consulta:

```
INSERT INTO películas  
SELECT MAX(codigo)+1, 'ESDLA', 180, 2001, 'Peter Jackson', 'Fantasía', 1  
FROM películas
```

- Esta sentencia tiene la ventaja de que funciona tanto en Oracle como en SQL Server y MariaDB, tal y como está, y en Access si añadimos tras INTO películas unos paréntesis y el nombre de las columnas de destino de los valores.

Actividad 2

Crea las querys necesarias para lograr lo siguiente:

- 1. Insertar un equipo de tu elección con valores específicos*
- 2. Insertar varios jugadores de ese equipo (en la misma consulta)*
- 3. Insertar jugadores en una tabla utilizando una subconsulta*
- 4. Insertar un equipo en una tabla utilizando la cláusula 'DEFAULT VALUES'*
- 5. Guardar en la tabla FutbolistasEspanha todos los futbolistas que jueguen para equipos españoles y separar el nombre de apellido*

15 MINUTOS

ACTUALIZACIÓN DE DATOS

Actualización de datos

Modificación de datos

- A la hora de analizar la información que se almacena en las tablas de una base de datos se podría distinguir dos tipos:
 - **Información que se puede considerar invariable** (en nuestra gamoteca podría ser, por ejemplo, el nombre y apellidos de un socio y el título y el creador de un juego)
 - **Información susceptible de sufrir cambios en el tiempo** (en nuestra gamoteca podría ser, por ejemplo, la dirección de un socio, la disponibilidad de un juego o la fecha en la que fue prestado por última vez)

Actualización de datos

- En cualquiera de los dos tipos anteriores, incluso en los considerados invariables ya que podría haberse introducido algún dato por error, puede ser necesario realizar alguna modificación. Para realizarla tendremos que recurrir a la cláusula `UPDATE`, que forma parte del lenguaje de manipulación de datos de SQL.
- Su sintaxis es la siguiente:

```
UPDATE tabla  
SET col1=valor1,col2=valor2...  
[WHERE condición]
```

Actualización de datos

- Aunque la cláusula `WHERE` es opcional, hay que tener mucho cuidado de no incluirla puesto que de no hacerlo, la modificación propuesta se aplicaría a todas las filas de la tabla. La finalidad de la cláusula `WHERE` es seleccionar las filas a cuyas columnas se asignarán los valores indicados por `SET`.

Actualización de datos

Cambiar una columna de una fila

- Suele ser muy habitual el uso de la sentencia UPDATE cuando necesitamos cambiar el dato contenido en una columna de una fila determinada. Por ejemplo, en nuestro caso, actualizar la columna disponible de un juego que acaba de ser prestado o devuelto. En este caso la sintaxis sería:

```
UPDATE tabla  
SET columna=valor  
WHERE claveprimaria=valor;
```

Actualización de datos

- Vamos a probar esta nueva cláusula en nuestra gamoteca. Para ello vamos a modificar el estado disponible de uno de nuestro juegos. Comprobemos antes de nada cuáles están disponibles y cuáles no.



```
SELECT codigo, titulo, disponible  
FROM juegos
```

196 %

Resultados Mensajes

	codigo	titulo	disponible
1	1	Final Fantasy VII	S
2	2	Diablo II	S
3	3	Portal 2	S
4	4	Half-Life	N
5	5	Day of the Tentacle	S

Actualización de datos

- Supongamos que el socio que tenía en préstamo el juego BioShock acaba de devolverlo. En este caso la columna disponible de dicho juego debe ser modificada pasando de N a S.
- Para ello ejecutamos la siguiente sentencia:

```
UPDATE juegos  
SET disponible = 'S'  
WHERE codigo = 7;
```

Actualización de datos

Cambiar varias columnas de una fila

- Si queremos modificar varias columnas en la misma sentencia, ésta será idéntica a la utilizada en el ejemplo anterior simplemente añadiéndole después de la cláusula SET las parejas de nombres columna-valor separados por comas:

```
UPDATE tabla  
SET columna1=valor1,columna2=valor2,...  
WHERE claveprimaria=valor;
```

Actualización de datos

- Como ejemplo vamos a modificar en nuestra `gameteca` los datos de dirección y código postal de un socio. Para identificar la fila vamos a utilizar el `nif` que es la clave primaria de la tabla.
- En la tabla anterior vamos a modificar los datos, de dirección y código postal, del socio Pedro Alonso Regojo.

Actualización de datos

- Consultemos de nuevo la tabla socios para verificar si se ha realizado la modificación.

```
UPDATE socios  
SET direccion = 'Abu Dhabi', cp = '36860'  
WHERE nif='12346578Z'
```


Actualización de datos

Modificación de datos en varias filas

- Si necesitamos asignar el mismo valor a las mismas columnas de varias filas, no es preciso que utilicemos una sentencia `UPDATE` individual para cada fila utilizando la clave principal como referencia de selección. En su lugar, sustituiremos el filtro de búsqueda de la cláusula `WHERE` por uno adecuado que nos permita actuar sobre las filas que nos interesen.
- Para asegurarnos que de que vamos a manipular las filas adecuadas podemos ejecutar una consulta previa seleccionando las filas sobre las que pretendemos actuar en la actualización.

Actualización de datos

- Como ejemplo en nuestra `gameteca`, vamos a actualizar la columna dirección de la tabla socios para todos aquellos en los que dicha columna tenga el valor de `NULL`, poniéndole el valor de Desconocida.

Actualización de datos

- Tras la última consulta hemos visto que se han seleccionado justamente las filas que pretendo actualizar por lo que procedemos a ello:

```
UPDATE socios  
SET direccion = 'Desconocida'  
WHERE direccion IS NULL
```

Actualización de datos

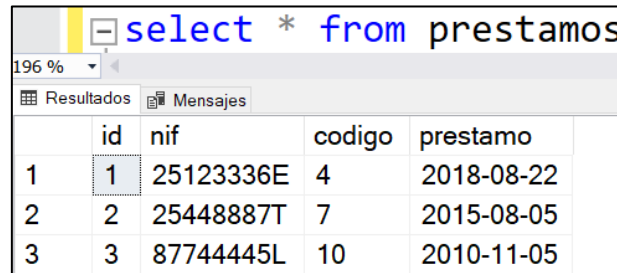
Uso de expresiones en la asignación

- Hasta ahora hemos modificado las columnas asignando un valor concreto a cada una. También podemos asignar valores con expresiones con funciones SQL.
- Supongamos, por ejemplo, que queremos extender la fecha de préstamo de un juego otros 15 días. La sentencia SQL sería:

```
UPDATE prestamos  
SET prestamo = prestamo + DATEADD(DAY,15,prestamo)
```

Actualización de datos

- Otra posibilidad interesante es basar la actualización de los datos de una tabla en los resultados de una consulta sobre otra tabla. Para entender mejor esta opción nos vamos a apoyar en un ejemplo.
- La tabla prestamos, que registra el préstamo de juegos, presenta el siguiente estado:



```
select * from prestamos
```

196 %

Resultados Mensajes

	id	nif	codigo	prestamo
1	1	25123336E	4	2018-08-22
2	2	25448887T	7	2015-08-05
3	3	87744445L	10	2010-11-05

Actualización de datos

- Hay tres juegos en préstamo, tres títulos que en su columna disponible contendrán el valor N, como podemos comprobar a continuación:

select * from juegos

	codigo	signatura	titulo	autor	disponible
1	1	I F final	Final Fantasy VII	Square Enix	S
2	2	I D Diablo	Diablo II	Blizzard	S
3	3	I P port	Portal 2	Valvei	S
4	4	I H hal	Half-Life	Valve Entertainment	N
5	5	I D day	Day of the Tentacle	Lucas Arts	S
6	6	T H half	Half-Life 2	Valvei	S
7	7	T B bio	BioShock	2K Games	N
8	8	T T the	The Witcher	CD Project	S
9	9	G T walk	The Walking Dead	Telltale Games	S
10	10	G E ense	Age of Empires	Ensemble Estudios	N
11	11	G T elder	The Elder Scrolls	Bethesda Game Studios	Sin info
12	12	G T ark	Lost Ark	Amazon	S
13	13	G T destiny	Destiny	Bungie	Sin info
14	14	G T pokemon	Pokemon: Arceus	Game Freak	Sin info
15	15	G T fifa	FIFA 22	EA Sports	Sin info
16	16	G T tsushima	Ghost of Tsushima	Sucker Punch	Sin info
17	17	G T last	Last of Us 2	Naughty Dog	Sin info
18	18	G T last1	Last Of Us	Naughty Dog	Sin info
19	19	G T god	God of War	Santa Monica Studios	Sin info
20	20	I F mario	Mario Kart	Nintendo	Sin info
21	21	g ava HP	Hogwarts Legacy	Avalanche	S

Actualización de datos

- Supongamos que dos socios se llevan sendos juegos en préstamo, algo que actualizamos agregando dos nuevas filas en la tabla `prestamos`:

```
INSERT INTO prestamos  
VALUES('12346578Z',1, CURRENT_TIMESTAMP)
```

```
INSERT INTO prestamos  
VALUES('12346578Z',5, CURRENT_TIMESTAMP)
```

- En este momento hay dos juegos que no están disponibles, puesto que acaban de ser prestados, pero su columna disponible, en la tabla `juegos`, sigue conteniendo el valor 'S'.

Actualización de datos

- Para que la base de datos fuese coherente tendría que actualizar dicha columna, por ejemplo, con sendas sentencias `UPDATE`. El problema se tendría si en lugar de dos, tengo que actualizar muchos juegos.
- Resultará mucho más cómodo utilizar un sentencia como la siguiente:

```
UPDATE juegos  
SET disponible = 'N'  
WHERE codigo IN  
(SELECT codigo from prestamos)
```


Actualización de datos

- En lugar de aplicar el proceso a todos los juegos prestados, podríamos añadir una cláusula `WHERE` a la subconsulta de la cláusula `IN` para recuperar sólo las filas de los prestamos correspondientes al día en curso.

```
UPDATE juegos
SET disponible = 'N'
WHERE codigo IN
(SELECT codigo from prestamos WHERE prestamo = CURRENT_TIMESTAMP)
```

Actualización de datos

- En cualquier caso, ahora los juegos contendrán el valor adecuado en la columna disponible, sin necesidad de efectuar actualizaciones individuales. Además las tablas serán coherentes:

`SELECT * FROM JUEGOS`

196 %

Resultados Mensajes

	codigo	signatura	titulo	autor	disponible
1	1	I F final	Final Fantasy VII	Square Enix	N
2	2	I D Diablo	Diablo II	Blizzard	S
3	3	I P port	Portal 2	Valvei	S
4	4	I H hal	Half-Life	Valve Entertainment	N
5	5	I D day	Day of the Tentacle	Lucas Arts	N
6	6	T H half	Half-Life 2	Valvei	S
7	7	T B bio	BioShock	2K Games	N
8	8	T T the	The Witcher	CD Project	S
9	9	G T walk	The Walking Dead	Telltale Games	S
10	10	G E ense	Age of Empires	Ensemble Estudios	N

`SELECT * FROM prestamos`

196 %

Resultados Mensajes

	id	nif	codigo	prestamo
1	1	25123336E	4	2018-08-22
2	2	25448887T	7	2015-08-05
3	3	87744445L	10	2010-11-05
4	4	12346578Z	1	2023-02-14
5	5	12346578Z	5	2023-02-14

Actualización de datos

Valores nulos y por defecto

- Con la sentencia `UPDATE` también se puede utilizar las palabras clave `NULL` y `DEFAULT` para introducir en una columna el valor nulo y su valor por defecto, respectivamente.
- Hay que recordar que el valor **`NULL`** representa la ausencia de contenido, lo cual no es equivalente a una cadena de caracteres vacía, el valor 0 o cualquier otro valor. El intento de asignar un valor nulo a un columna podría provocar un error, en caso de en la definición de la tabla se indicase que esa columna no puede quedar vacía.
- El uso de **`DEFAULT`** tendrá sentido sólo cuando durante la definición de la estructura de la tabla, se hubiese establecido un valor por defecto para las columnas.

Actividad 3

Crea las queries necesarias para lograr lo siguiente:

1. Actualizar los puntos de un equipo en La Liga.
2. Actualizar el valor del precio de mercado de un jugador en un equipo.
3. Actualizar los partidos jugados, ganados, perdidos y empatados por un equipo.
4. Realizar el traspaso de un futbolista a otro equipo.
5. Hacer un intercambio de futbolistas, pero manteniendo las estadísticas actuales (Jugador1 va a Equipo2, y Jugador2 va a Equipo1)
6. Retirar a un futbolista (Cambiar su dorsal a 0 y su equipo a 'RETIRADO')
7. Mudar a un equipo de país, con todo los cambios que eso conlleva (empieza de 0 en partidos, goles, etc.)

15 MINUTOS

Actualización de datos

Eliminación de filas

- Para eliminar una fila se utiliza la cláusula `DELETE`, que actúa sobre una fila completa según la siguiente sintaxis:

```
DELETE FROM tabla  
[WHERE condición];
```

- Aunque es opcional, la cláusula `WHERE` resulta imprescindible a la hora de ejecutar una operación de eliminación de filas ya que, de lo contrario, estaríamos borrando todo el contenido actual de la tabla.
- Si ejecutamos la sentencia **`DELETE FROM tabla`**, la estructura de la tabla permanecería de tal forma que sería posible añadir nuevas filas, pero todas las que hubiese hasta ese momento se perderían.

Actualización de datos

- Las dos formas de eliminar filas son:
 - **Borrado de una fila concreta**, utilizando en la cláusula `WHERE` el nombre de la columna que actúa como clave primaria y el valor que seleccionará es fila de manera única
 - **Eliminación de varias filas**, usando para ello un filtro de selección adecuado
- Al igual que ocurría con la cláusula `UPDATE`, también con `DELETE` resulta recomendable, cuando vayamos a eliminar varias filas, primero realizar una consulta utilizando la cláusula `SELECT` con el filtro `WHERE` adecuado, de modo que tengamos la certeza de que dicho filtro selecciona las filas exactas que posteriormente se van a eliminar.

Actualización de datos

- Para probar esta cláusula vamos a eliminar en la tabla socios de nuestra gamoteca, todos los socios que han sido dados de alta el día 30/12/2018.

```
DELETE FROM socios  
WHERE alta = '2018-12-30'
```

Actividad 4

Crea las queries necesarias para lograr lo siguiente:

- 1. Eliminar un jugador de un equipo.*
- 2. Eliminar los equipos italianos.*
- 3. Eliminar un equipo de La Liga, y por lo tanto, a todos sus jugadores.*
- 4. Eliminar a todos los porteros de los equipos.*
- 5. Eliminar a todos los canteranos de los equipos (no tienen dorsal entre 1 y 25)*
- 6. Eliminar a aquellos equipos que tengan menos de 23 jugadores (y, por supuesto, a sus jugadores)*
- 7. Eliminar a aquellos jugadores que se hayan retirado (o pasen de 40 años)*

Actualización de datos

Fusión de datos (MERGE)

- Hay situaciones en las que nos puede interesar llevar a cabo una operación u otra sobre una tabla dependiendo de la información existente en otra tabla o del resultado obtenido de una consulta, fusionando datos de acuerdo a ciertas reglas. Esto se consigue con la sentencia `MERGE` que no está presente en MariaDB, pero si lo está en Oracle y SQL Server.
- La sintaxis sería:

Actualización de datos

```
MERGE INTO tabladestino  
USING tablaorigen | (consulta)  
ON (predicado)  
WHEN MATCHED THEN  
UPDATE | DELETE  
WHEN NOT MATCHED THEN  
INSERT
```

Actualización de datos

- En la sentencia anterior:
 - **tabladestino**, es aquella sobre la que se van a realizar las operaciones de actualización, que pueden ser actualización de los datos, borrado de la fila coincidente o bien la inserción de una nueva fila
 - Después de la cláusula **ON** hay que especificar un predicado de emparejamiento de filas entre **tabladestino** y **tablaorigen** (o la consulta)
 - Apartado **WHEN MATCHED** da paso a la operación a efectuar para cada fila coincidente, que será actualizar los valores en la tabla de destino o bien eliminar la fila
 - Apartado **WHEN NO MATCHED** da paso a la operación a realizar para cada fila no coincidente, que sólo puede ser una inserción

Actualización de datos

- Para entender esta sentencia supongamos que en nuestra gamoteca la tabla `prestamos` recoge los préstamos diarios, tras lo cual su contenido se fusiona con otras tablas y después se borra. Por lógica la tabla destinataria de la fusión sería la de `juegos`, actualizando adecuadamente su columna `disponible`. Otra posibilidad es que se hayan prestado juegos nuevos, a los que se les ha asignado un código pero aún no han sido registrados en la tabla `juegos`.
- Nuestra sentencia de fusión de la tabla `prestamos` en la tabla `juegos` sería:

Actualización de datos

```
MERGE INTO juegos
USING prestamos
ON (juegos.codigo=prestamos.codigo)
WHEN MATCHED THEN
UPDATE SET juegos.disponible='N'
WHEN NOT MATCHED THEN
INSERT (codigo, signature, titulo)
VALUES (prestamos.codigo, 'D', 'No registrado');
```

Actualización de datos

- La sentencia SQL anterior busca en la tabla `juegos` aquellas filas cuyo código coincide con los códigos almacenados en la tabla `prestamos`. Por cada fila de esta última tabla se realizará una de dos operaciones posibles:
 - Si en la tabla `juegos` existe un juego con el código correspondiente, se actualizará su columna `disponible` dándole el valor 'N'.
 - Si en la tabla `juegos` no existe un juego con dicho código, se añadirá una fila nueva con el código apropiado pero el resto de los datos pendientes de introducir.

Actualización de datos

- En el ejemplo suponemos que todas las filas corresponden a nuevos préstamos, pero podría sustituirse la cláusula `USING prestamos` por una del tipo `USING (SELECT col FROM prestamos WHERE...)` para obtener las filas de préstamos por una parte y las de devolución por otra, dando a la columna `disponible` el valor en cada caso.

Actividad 5

Crea las querys necesarias para lograr lo siguiente:

- 1. Actualizar los datos de un jugador, si el jugador ya existe, o insertarlo si es un jugador nuevo.*
- 2. Actualizar los datos de un equipo, si el equipo ya existe, o insertarlo si es un equipo nuevo.*
- 3. Actualizar los datos de un defensa, si el defensa ya existe, o insertarlo si es un defensa nuevo.*
- 4. Borrar los datos de un equipo, si el equipo ya existe, o insertarlo si es un equipo nuevo.*

10 MINUTOS

Actualización de datos

Actualización de datos y transacciones

- Tanto la inserción de datos, la modificación y eliminación son operaciones que se verán afectadas por las transacciones en curso que pudieran existir en el RDBMS.
- Ante operaciones potencialmente peligrosas, como la sustitución de datos por otros o la eliminación de datos, su inclusión en el contexto de una transacción puede ahorrarnos un serio disgusto.
- La sentencia a usar para iniciar una transacción es **START TRANSACTION**.
- Ésta no se reconoce en Oracle ya que en él las transacciones se inician automáticamente. En SQL Server y Access tendremos que usar **BEGIN TRANSACTION**

Actualización de datos

- **COMMIT:** Confirma una operación que hayamos efectuado en base de datos.
- **ROLLBACK:** Rechaza las transacciones pendientes, deshaciendo los cambios que hubiese sufrido la base de datos
- Para entender esto supongamos que queremos eliminar de la tabla prestamos una serie de filas correspondientes a los juegos entregados hoy por los socios de la gamoteca usando para ello una sentencia DELETE con un criterio de selección asociado. Sin embargo, al escribir la sentencia escribimos DELETE FROM prestamos, olvidándonos de la cláusula WHERE y provocando que todas las filas de la tabla prestamos se pierdan. Nada habría ocurrido si con anterioridad hubiese iniciado una transacción.

Transacciones en SQL Server

- A continuación, veremos un vídeo explicando las transacciones:

<https://youtu.be/xiTfest5ApU?si=PYAWngtj058aVJtp>

Actividad 6

Crea las querys necesarias para lograr lo siguiente:

- 1. Realizar una transacción para agregar un nuevo jugador y actualizar la lista de goles de un equipo con los goles de ese jugador*
- 2. Realizar una transacción para mover a un jugador de un equipo y no restar la cantidad de goles pero sí sumársela al nuevo equipo.*
- 3. Realizar una transacción para agregar un nuevo equipo y agregar 2 jugadores nuevos a ese equipo.*

15 MINUTOS



**KEEP
CALM
IT'S
KAHOOT
TIME**

Tema 3: El lenguaje SQL (II)

Asignatura: Bases de datos

CS Desarrollo de Aplicaciones Web

