1. Los verbos HTTP, también conocidos como métodos HTTP, son acciones que se utilizan en las solicitudes para indicar qué tipo de operación se debe realizar en un recurso en un servicio web

En el caso de API REST. En el contexto de una API REST, estos verbos son fundamentales para realizar diferentes tipos de operaciones como el GET, POST, PUT, DELETE, PATCH

GET: se utiliza para obtener información del servidor, como obtener datos

POST: se utiliza para enviar datos nuevos al servidor, ejemplo: para insertar datos a una base de datos o la creación de un nuevo recurso

PUT: se utiliza para actualizar datos o recursos en el servidor, ejemplo: cuando actualizo datos de un registro de base de datos teniendo en cuenta que el recurso ya existe

PATCH: es muy similar a un PUT, pero con la diferencia de que actualizo recursos específicos, ejemplo: cuando quiero actualizar un registro específico de una tabla en base de datos sin actualizar todo el objeto en si.

1. Explicar que hace este método
2. **internal** static **void** SingleToMulti(int[] array, int row, int column)
3. {
4. int index = 0;
5. int[,] multi = **new** int[row, column];
6. **for** (int y = 0; y < row; y++)
7. {
8. **for** (int x = 0; x < column; x++)
9. {
10. multi[y, x] = array[index];
11. index++;
12. Console.Write(multi[y, x] + " ");
13. }
14. Console.WriteLine();
15. }
16. }

Este método define un método llamado SingleToMulti que convierte un arreglo unidimensional en una matriz bidimensional donde tiene 3 argumentos, luego un arreglo unidimensional “array”, las filas “row” y columnas “column”. Luego se utiliza una variable llamada Index que inicializa y sirve para rastrear la posición actual del arreglo unidimensional “int[,] multi = new int[row, column];”

Se emplea un for anidado donde el primero se inicia para iterar sobre las filas, y el segundo for itera sobre las columnas dentro de la fila actual. Y dentro del cuerpo del bucle anidado “multi[y, x] = array[index]” se signa el valor del elemento actual del arreglo unidimensional al elemento correspondiente en la matriz bidimensional utilizando las variables y y x como índices de fila y columna respectivamente.

index++;: Incrementa el índice para apuntar al siguiente elemento en el arreglo unidimensional.

Y por último se imprime el resultado que es el valor actual de la matriz bidimensional, como se aprecia a continuación “**Console.Write(multi[y, x] + " ");**”

3) a- Cree un diagrama de relación de entidad para describir la consulta. Debe incluir entidades, sus relaciones, claves primarias y tipos de datos.

Entidades:

T\_TSERVICE\_REQUEST (Entidad principal para las solicitudes de servicio)

Atributos: SR\_ID (Clave primaria), Requestor\_Mail, Requestor\_Name, TS\_Created, TS\_Assigned, TS\_Completed, TS\_Closed, SRStatus, Customer Name, CBC\_ID, DealOwnerID, DealOwnerFunction, TS\_TierCommunicated, TierCommunicator, SRCompletionCommunicator, Country, Area, ExpectedDurationHH.

T\_USERS (Usuarios)

Atributos: NT\_User, Name.

T\_FUNCTION (Funciones)

Atributos: FunctionID, FunctionDescription.

T\_TR\_INBOX (Bandeja de entrada)

Atributos: Inbox\_ID, TS\_ReceivedExchange.

T\_TSR\_CIC\_INFO (Información de CIC para las solicitudes de servicio)

Atributos: SR\_ID, Field\_XName, FieldValue.

Relaciones:

T\_TSERVICE\_REQUEST se relaciona con T\_USERS mediante DealOwnerID (clave foránea).

T\_TSERVICE\_REQUEST se relaciona con T\_FUNCTION mediante DealOwnerFunction (clave foránea).

T\_TSERVICE\_REQUEST se relaciona con T\_TR\_INBOX mediante InboxID (clave foránea).

T\_TSERVICE\_REQUEST se relaciona con T\_TSR\_CIC\_INFO mediante SR\_ID (clave foránea).

Claves primarias:

T\_TSERVICE\_REQUEST: SR\_ID

T\_USERS: NT\_User

T\_FUNCTION: FunctionID

T\_TR\_INBOX: Inbox\_ID

T\_TSR\_CIC\_INFO: (Combinación de SR\_ID y Field\_XName)

b- ¿Cuál es el resultado si cambiamos todas las relaciones a LEFT JOIN en lugar de INNER JOIN?

Si se cambian todas las relaciones de INNER JOIN a LEFT JOIN en la consulta, tendríamos un resultado diferente en el sentido de que la consulta también incluiría registros que no coinciden en las tablas relacionadas. Es decir, los registros de la tabla principal (en este caso, T\_TSERVICE\_REQUEST) se mostrarían incluso si no hay coincidencias en las tablas relacionadas (T\_USERS, T\_FUNCTION, T\_TR\_INBOX, T\_TSR\_CIC\_INFO).

C - En tus propias palabras ¿cuál es la operación que realiza la función F\_FUNC?

La función F\_FUNC calcula el número de días hábiles entre dos fechas, excluyendo los fines de semana y ajustando si el inicio o el final caen en un día no laborable (domingo o sábado).

Es decir, dado un período de tiempo entre dos fechas, la función calcula cuántos días de trabajo (laborables) hay en ese período, descontando los días no laborables (sábados y domingos) y considerando si las fechas de inicio y final también son días no laborables.

4) Se le proporciona una matriz NxN en blanco, siendo N un número impar. Tu propósito es llenar el cuadrado según las siguientes reglas:

1. Siempre comienzas con el número 1, y siempre comenzará en la fila superior y en la columna del medio (es decir, si la matriz es 3x3, el número uno comenzará en la fila 1 y la columna 2).

a)

using System;

class Program

{

static void Main(string[] args)

{

int n = 5; // Tamaño de la matriz (debe ser impar)

int[,] matrix = new int[n, n];

FillMatrix(matrix);

PrintMatrix(matrix);

}

static void FillMatrix(int[,] matrix)

{

int num = 1;

int row = 0;

int col = matrix.GetLength(1) / 2;

while (num <= matrix.GetLength(0) \* matrix.GetLength(1))

{

matrix[row, col] = num;

num++;

int newRow = (row - 1 + matrix.GetLength(0)) % matrix.GetLength(0);

int newCol = (col + 1) % matrix.GetLength(1);

if (matrix[newRow, newCol] == 0)

{

row = newRow;

col = newCol;

}

else

{

row = (row + 1) % matrix.GetLength(0);

}

}

}

static void PrintMatrix(int[,] matrix)

{

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

Console.Write(matrix[i, j].ToString().PadLeft(4));

}

Console.WriteLine();

}

}

}

b)

El siguiente número estará una fila hacia arriba y una columna a la derecha. Si no hay filas arriba, use la de abajo, si no hay más columnas a la derecha, use la de arriba a la izquierda.

using System;

class Program

{

static void Main(string[] args)

{

int n = 5; // Tamaño de la matriz (debe ser impar)

int[,] matrix = new int[n, n];

FillMatrix(matrix);

PrintMatrix(matrix);

}

static void FillMatrix(int[,] matrix)

{

int num = 1;

int row = 0;

int col = matrix.GetLength(1) / 2;

while (num <= matrix.GetLength(0) \* matrix.GetLength(1))

{

matrix[row, col] = num;

num++;

int newRow = (row - 1 + matrix.GetLength(0)) % matrix.GetLength(0);

int newCol = (col + 1) % matrix.GetLength(1);

if (matrix[newRow, newCol] == 0)

{

row = newRow;

col = newCol;

}

else

{

row = (row + 1) % matrix.GetLength(0);

}

}

}

static void PrintMatrix(int[,] matrix)

{

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

Console.Write(matrix[i, j].ToString().PadLeft(4));

}

Console.WriteLine();

}

}

}

5)

Modelo ER:

Entidades:

Proveedor:

Atributos: ID (PK), Nombre

Producto:

Atributos: ID (PK), Nombre

Región:

Atributos: ID (PK), Nombre

TipoCuenta:

Atributos: ID (PK), Nombre (Cuenta Global, Cuenta Local)

Venta:

Atributos: ID (PK), ProveedorID (FK), ProductoID (FK), RegionID (FK), TipoCuentaID (FK), Hardware, Software, Services, SalesPerson

Procedimientos almacenados

SP\_VentasCuentasGlobales:

CREATE PROCEDURE SP\_VentasCuentasGlobales

AS

BEGIN

SELECT SUM(Hardware) AS TotalHardware, SUM(Software) AS TotalSoftware

FROM Venta

WHERE TipoCuenta = 'Cuenta Global';

END

SP\_VentasPorRegion:

CREATE PROCEDURE SP\_VentasPorRegion

AS

BEGIN

SELECT Region, SUM(Hardware + Software + Services) AS TotalVentas

FROM Venta

GROUP BY Region;

END

SP\_VentasPorTipoProducto:

CREATE PROCEDURE SP\_VentasPorTipoProducto

AS

BEGIN

SELECT Producto, SUM(Hardware) AS TotalHardware, SUM(Software) AS TotalSoftware, SUM(Services) AS TotalServices

FROM Venta

GROUP BY Producto;

END