Causa raíz notas de crédito no timbradas

Autor: Huilver Nolasco Aguilar

Fecha: Enero del 2025

Contenido

Descripción general

Se detectó que desde finales del año 2023 no se han timbrado notas de crédito tipo ‘ND’ ya que no se detectó ni el uuid ni la fecha de timbrado, una descripción más amplia del problema se describe en Notas de Credito.docx

Causa

Para abordar la revisión se analizaron los diferentes logs del central, sin embargo, no se detectó realmente nada concluyente, pero sí indicios para formular una conjetura por validar.

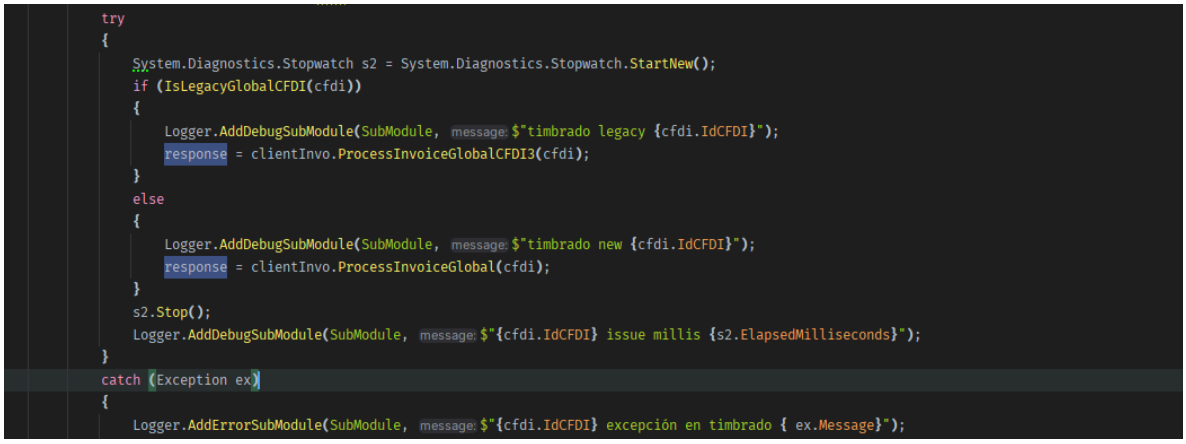
Dentro de lo que se observó en los logs fue que no había ningún mensaje acorde al código debía estar pintando, solo mensajes del tipo:

```
[2025-01-15 23:59:57:157],FX_CENTRAL,GLOBALCFDI.ISSUE,INFO,,11003730020250115 - legacy False milis 6144 message
[2025-01-15 23:59:57:554],FX_CENTRAL,GLOBALCFDI.ISSUE,INFO,,11003770020250115 - legacy False milis 6338 message
[2025-01-15 23:59:58:389],FX_CENTRAL,GLOBALCFDI.ISSUE,INFO,,11004300020250115 - legacy False milis 6104 message
[2025-01-15 23:59:58:812],FX_CENTRAL,GLOBALCFDI.ISSUE,INFO,,11004370020250115 - legacy False milis 6376 message
[2025-01-15 23:59:59:609],FX_CENTRAL,GLOBALCFDI.ISSUE,INFO,,11003260020250115 - legacy False milis 6098 message
```

y que indica que esos registros si se enviaron a timbrar ya que corresponde con esta parte del código que lo demuestra

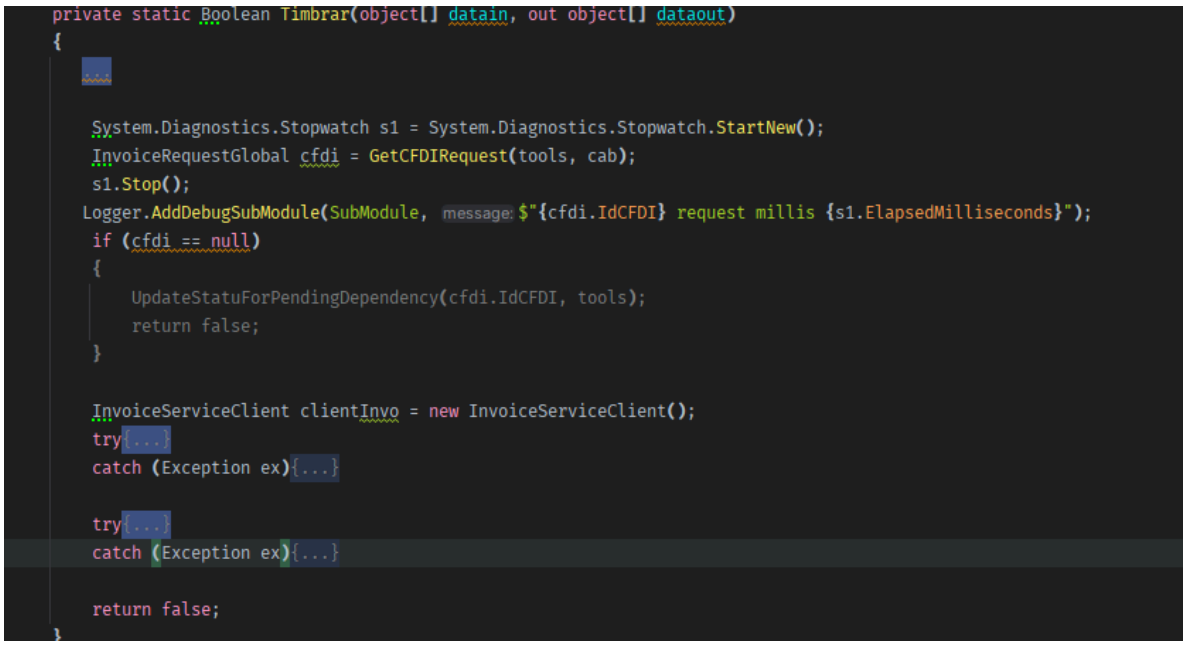
```
InvoiceServiceClient clientInvo = new InvoiceServiceClient();
try
{
    System.Diagnostics.Stopwatch s2 = System.Diagnostics.Stopwatch.StartNew();
    if (IsLegacyGlobalCFDI(cfdi))
    {
        Logger.AddDebugSubModule(SubModule, message: $"timbrado legacy {cfdi.IdCFDI}");
        response = clientInvo.ProcessInvoiceGlobalCFDI3(cfdi);
    }
    else
    {
        Logger.AddDebugSubModule(SubModule, message: $"timbrado new {cfdi.IdCFDI}");
        response = clientInvo.ProcessInvoiceGlobal(cfdi);
    }
    s2.Stop();
    Logger.AddDebugSubModule(SubModule, message: $"{cfdi.IdCFDI} issue millis {s2.ElapsedMilliseconds}");
}
catch (Exception ex){...}

try
{
    if (testMode ?? true){...}
    else if (response != null)
    {
        System.Diagnostics.Stopwatch s3 = System.Diagnostics.Stopwatch.StartNew();
        if (UpdateSatu(cfdi.IdCFDI, response, tools))
        {
            var x :CommonDataBusinessError = response.BusinessError.FirstOrDefault();
            string data = x == null ? String.Empty : x.Description;
            Logger.AddInfoSubModule(SubModule, message: $"{cfdi.IdCFDI} - legacy {IsLegacyGlobalCFDI(cfdi)} milis {s.ElapsedMilliseconds} message {data}");
```

Por lo anterior podemos concluir que los registros no timbrados o tuvieron un error en el timbrado o que no llegaron a eso por alguna razón; sin embargo, no hay ningún registro de ERROR en log y el código indica que si imprime errores bajo estas circunstancias:



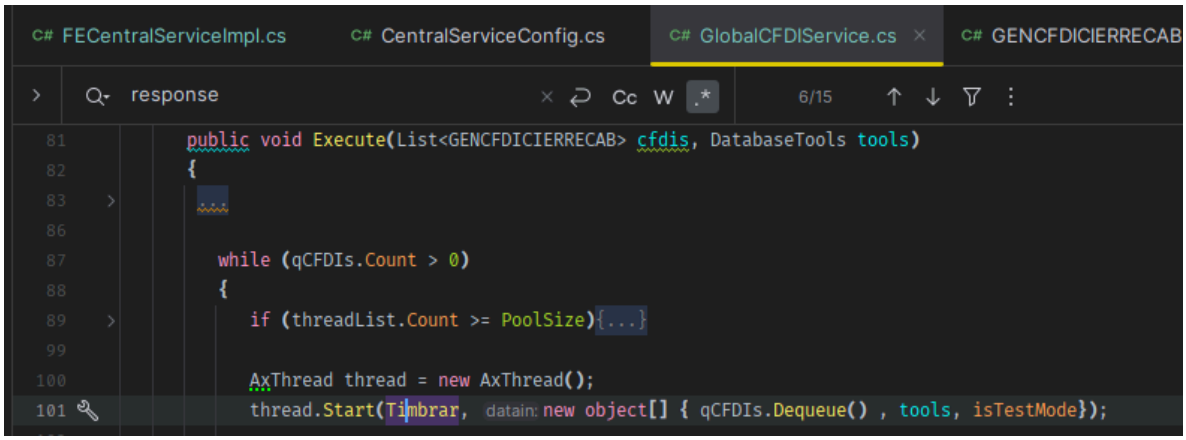
```
try
{
    System.Diagnostics.Stopwatch s2 = System.Diagnostics.Stopwatch.StartNew();
    if (IsLegacyGlobalCFDI(cfdi))
    {
        Logger.AddDebugSubModule(SubModule, message: $"timbrado legacy {cfdi.IdCFDI}");
        response = clientInvo.ProcessInvoiceGlobalCFDI3(cfdi);
    }
    else
    {
        Logger.AddDebugSubModule(SubModule, message: $"timbrado new {cfdi.IdCFDI}");
        response = clientInvo.ProcessInvoiceGlobal(cfdi);
    }
    s2.Stop();
    Logger.AddDebugSubModule(SubModule, message: $"{cfdi.IdCFDI} issue millis {s2.ElapsedMilliseconds}");
}
catch (Exception ex)
{
    Logger.AddErrorSubModule(SubModule, message: $"{cfdi.IdCFDI} excepción en timbrado { ex.Message}");
```

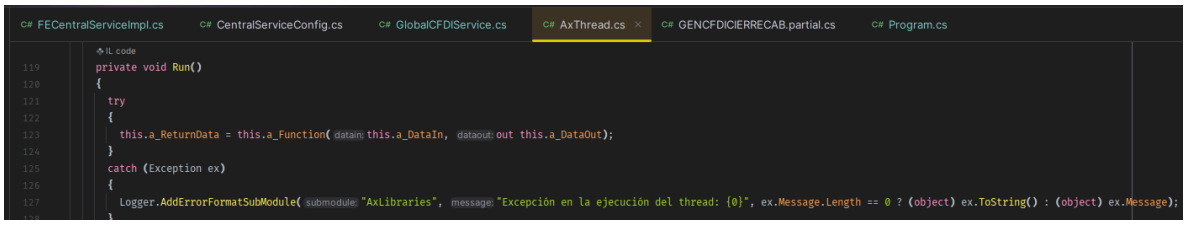
Por lo que entonces el problema viene de algún otro error diferente del timbrado, así que nos enfocamos a lo que ocurre previo al timbrado, que es en el método *GetCFDIRequest*, o en la impresión de logs dado que la validación *if (cfdi == null)* llega ahí si y sólo si el objeto cfdi es diferente de null, porque si es null se produce una excepción y como se puede ver en código no está manejada.



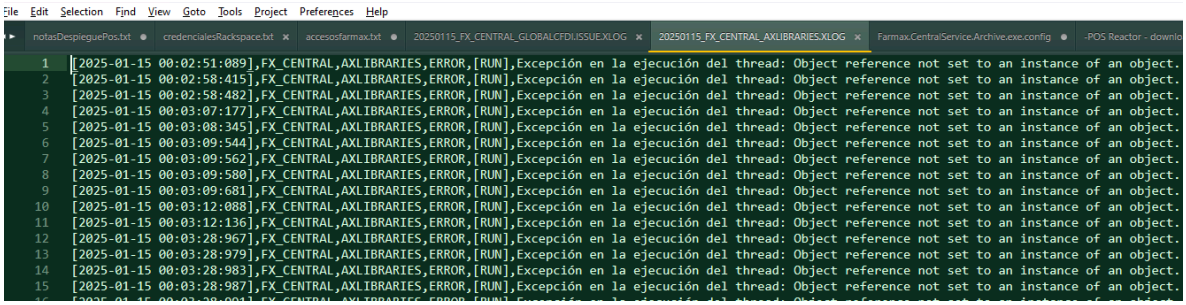
```
private static Boolean Timbrar(object[] datain, out object[] dataout)
{
    ...

    System.Diagnostics.Stopwatch s1 = System.Diagnostics.Stopwatch.StartNew();
    InvoiceRequestGlobal cfdi = GetCFDIRequest(tools, cab);
    s1.Stop();
    Logger.AddDebugSubModule(SubModule, message: $"{cfdi.IdCFDI} request millis {s1.ElapsedMilliseconds}");
    if (cfdi == null)
    {
        UpdateStatuForPendingDependency(cfdi.IdCFDI, tools);
        return false;
    }

    InvoiceServiceClient clientInvo = new InvoiceServiceClient();
    try{...}
    catch (Exception ex){...}

    try{...}
    catch (Exception ex){...}

    return false;
}
```

Pero si está manejada en el lugar donde se ejecuta el proceso

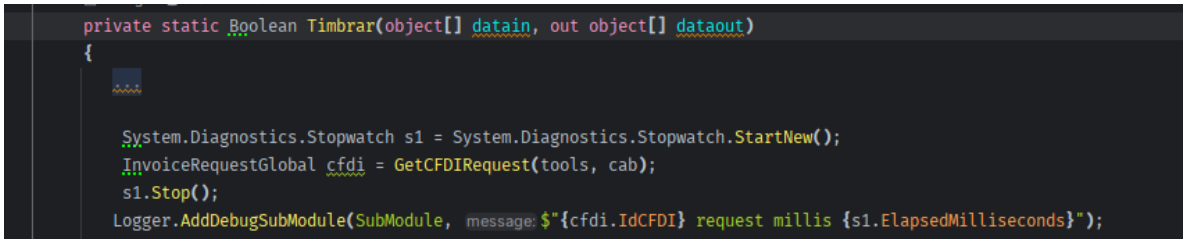




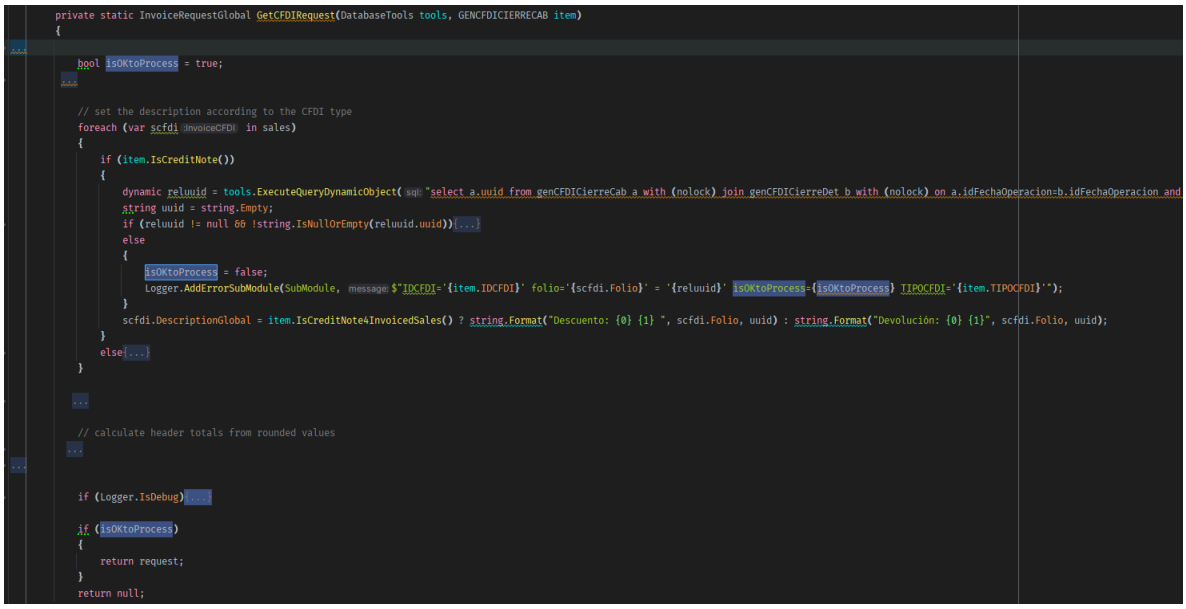
Por lo que buscamos en todos los logs y encontramos errores de este tipo en el log de AxLibraries, los cuales se encontraron y muchos:



Ahora si revisamos el código, este error se puede dar solo si el objeto *cfdi* es nulo y lanzando una excepción no controlada al tratar de imprimir *cfdi.IdCFDI* en el log



y eso se da porque la variable *isOKtoProcess* del siguiente código es *false*



```
private static InvoiceRequestGlobal GetCFDIRequest(DatabaseTools tools, GENCFDICIERRECAB item)
{
...
    bool isOKtoProcess = true;
...
    // set the description according to the CFDI type
    foreach (var scfdi in sales)
    {
        if (item.IsCreditNote())
        {
            dynamic reluuid = tools.ExecuteQueryDynamicObject( sql: "select a.uuid from genCFDICierreCab a with (nolock) join genCFDICierreDet b with (nolock) on a.idFechaOperacion=b.idFechaOperacion and
            string uuid = string.Empty;
            if (reluuid != null && !string.IsNullOrEmpty(reluuid.uuid))...
            else
            {
                isOKtoProcess = false;
                Logger.AddErrorSubModule(SubModule, message: $"IDCFDI='{item.IDCFDI}' folio='{scfdi.Folio}' = '{reluuid}' isOKtoProcess={isOKtoProcess} TIPOCFDI='{item.TIPOCFDI}'");
            }
            scfdi.DescriptionGlobal = item.IsCreditNote4InvoicedSales() ? string.Format("Descuento: {0} {1} ", scfdi.Folio, uuid) : string.Format("Devolución: {0} {1}", scfdi.Folio, uuid);
        }
        else...
    }
...
    // calculate header totals from rounded values
...
    if (Logger.IsDebug)...
    if (isOKtoProcess)
    {
        return request;
    }
    return null;
```

Por lo que si hay un camino que nos lleve a esos errores. Hasta aquí solo se sabe que si hay errores que están ocurriendo en ese proceso, sin embargo, aún no sabemos porqué sucede.

Para que la variable ***isOKToProcess*** se haga ***false*** se debe cumplir que el siguiente query *no traiga resultados o que si los trae el campo* ***uuid*** *venga* ***null*** es decir que lo siguiente **NO se cumpla**

reluuid != null && !string.IsNullOrEmpty(reluuid.uuid) *pero sobre todo que sea una nota de crédito.*

```
public const string TIPO_CFDI_CREDIT_NOTE_INVOICED = "NF";
public const string TIPO_CFDI_CREDIT_NOTE_RETURNED = "ND";


4 usages  Juan Murillo
public bool IsCreditNote()
{
    return this.IsCreditNote4InvoicedSales() || this.IsCreditNote4ReturnedSales();
}

2 usages  Juan Murillo
public bool IsCreditNote4InvoicedSales()
{
    return this.TIPOCFDI == TIPO_CFDI_CREDIT_NOTE_INVOICED;
}

1 usage  Juan Murillo
public bool IsCreditNote4ReturnedSales()
{
    return this.TIPOCFDI == TIPO_CFDI_CREDIT_NOTE_RETURNED;
}
```

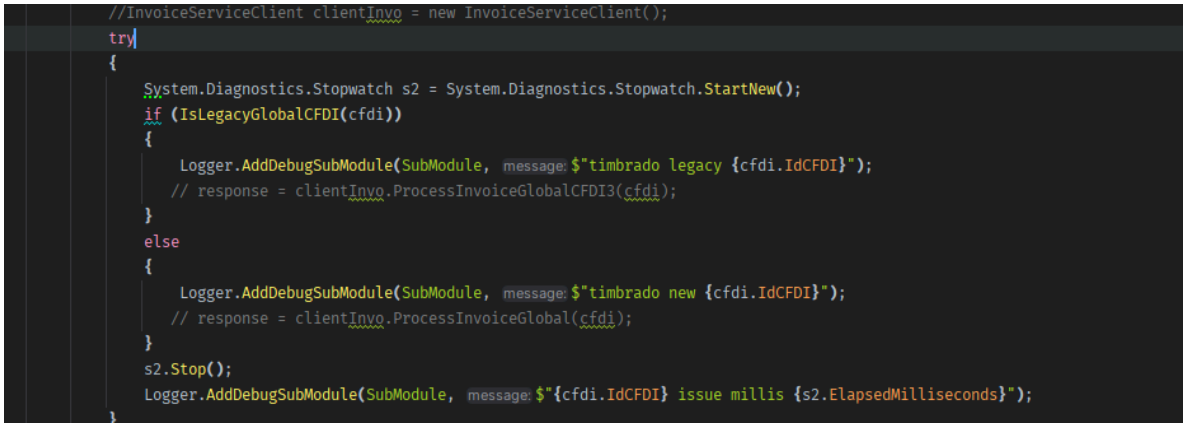
La consulta en cuestión pregunta si hay una venta devuelta en la nota de crédito que se pretende timbrar, y que la venta se hubiera reportado en una factura global.

```
select a.uuid from genCFDICierreCab a with (nolock) join genCFDICierreDet b with
(nolock) on a.idFechaOperacion=b.idFechaOperacion and a.idCFDI = b.idCFDI where
tipoCFDI='FG' and b.folio = ?
```

Dado que los logs no ofrecen información sobre qué notas de crédito están cayendo en este escenario, es necesario replicar el error pero ambientarlo no es una tarea fácil debido a que deben existir las facturas globales, las notas de crédito, etc., se decide replicarlo apuntando a archive con el debido cuidado de no impactar la data final y tampoco timbrar nada.

Replicación del error

Para lograr la replicación de error, nos ubicamos en el branch main que es el que corresponde al código productivo y nos ubicamos en la parte que realiza el timbrado y comentamos el que código realiza el timbrado para así evitarlo y no tener afectación



```
//InvoiceServiceClient clientInvo = new InvoiceServiceClient();
try
{
    System.Diagnostics.Stopwatch s2 = System.Diagnostics.Stopwatch.StartNew();
    if (IsLegacyGlobalCFDI(cfdi))
    {
        Logger.AddDebugSubModule(SubModule, message: $"timbrado legacy {cfdi.IdCFDI}");
        // response = clientInvo.ProcessInvoiceGlobalCFDI3(cfdi);
    }
    else
    {
        Logger.AddDebugSubModule(SubModule, message: $"timbrado new {cfdi.IdCFDI}");
        // response = clientInvo.ProcessInvoiceGlobal(cfdi);
    }
    s2.Stop();
    Logger.AddDebugSubModule(SubModule, message: $"{cfdi.IdCFDI} issue millis {s2.ElapsedMilliseconds}");
}
```

Además tomamos de producción las mismas variables que el proceso ocupa, en este caso:

```
<add key= "GLOBAL_CFDI_MAX_AGE_DAYS" value="40"/>
```

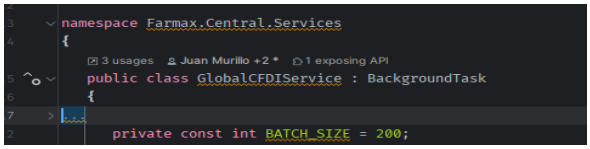
que se ocupa aquí:

```
string QUERY = $"SELECT top {BATCH_SIZE} a.idFechaOperacion, a.idCFDI, a.codigoSucursal, a.fechaOperacion, a.uuid,
a.tipoCFDI, a.montoInicial, a.montoDescuento, a.montoVendido, a.montoCuotasIEPS, a.montoTasasIEPS, a.montoIVA,
a.montoTotal, a.fechaGeneracion, a.fechaTimbrado, a.fechaEnvio, a.comprobanteFiscal, a.statu, a.syncCreadoFecha,
a.syncActualizadoFecha, a.syncActualizadoPor, a.syncVersionRegistro from genCFDICierreCab a where a.statu in ( 'R','D' )
and fechaOperacion> getdate() - { CentralServiceConfig.PendingCFDIMaxAge} order by cast(SUBSTRING(idcfdi,1,2)
as integer) + case statu when 'R' then 0 else 100 end asc";
```

dentro de este código:

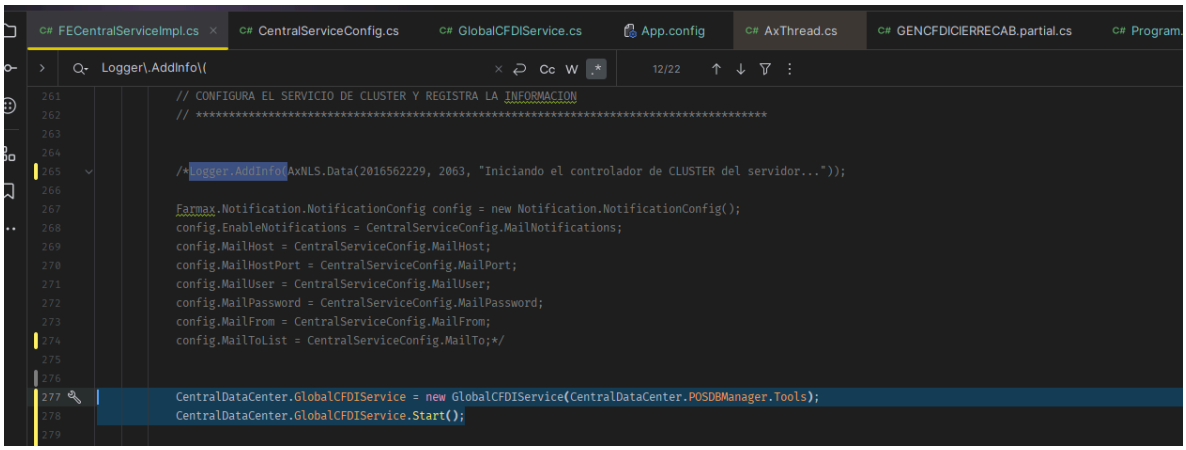
```
private bool SearchPendingInvoice(out List<GENCFDICIERRECAB> listCFDIs, DatabaseTools tools)
{
    ProcessResult rc = new ProcessResult();
    listCFDIs = new List<GENCFDICIERRECAB>();

    try
    {
        // order by a priority where received FGs are attempted first, deferred CFDIs are given lower priority
        SqlReportQuery queryObj = new SqlReportQuery();
        string QUERY = $"SELECT top {BATCH_SIZE} a.idFechaOperacion, a.idCFDI, a.codigoSucursal, a.fechaOperacion, a.uuid, a.tipoCFDI, a.montoInicial, a.montoDescuento, a.montoVendido, a.montoCuotasIEPS, a.montoTasasIEPS, a.montoIVA, a.montoTotal,
```

Y la constante BATCH_SIZE no es necesario configurarla debido a que viene gobernada en código

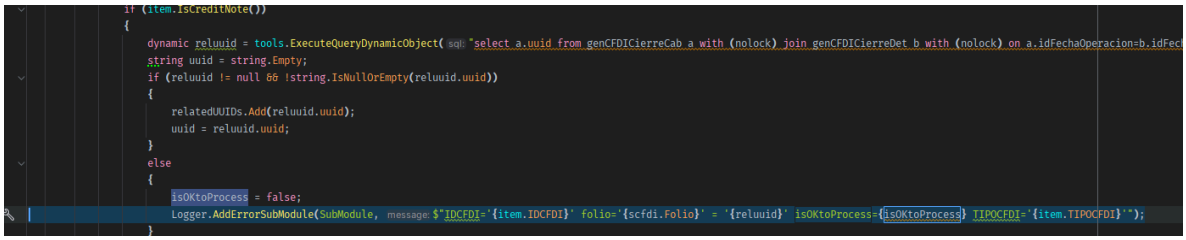


```
namespace Farmax.Central.Services
{
    3 usages  Juan Murillo +2 *  1 exposing API
    public class GlobalCFDIService : BackgroundTask
    {
        private const int BATCH_SIZE = 200;
```

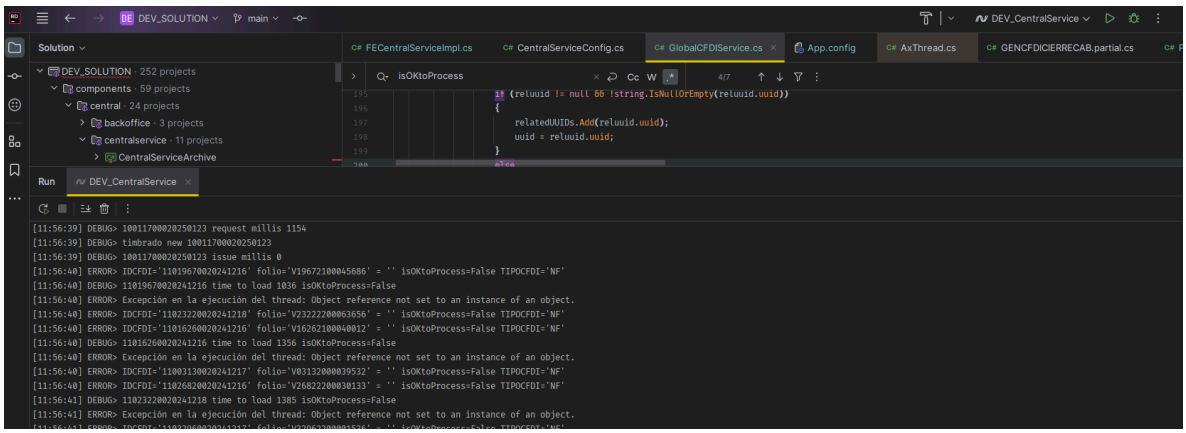
Además que solo dejamos activo el proceso de CFDI central para evitar tener los demás procesos corriendo.



Además agregamos un log que nos brinde información sobre la nota de crédito, su tipo y la venta que al validar hace que se produzca el error



Procedemos a ejecutarlo:



Como se puede observar se están mostrando en log los mismos errores que en el log de AxLibraries, es decir

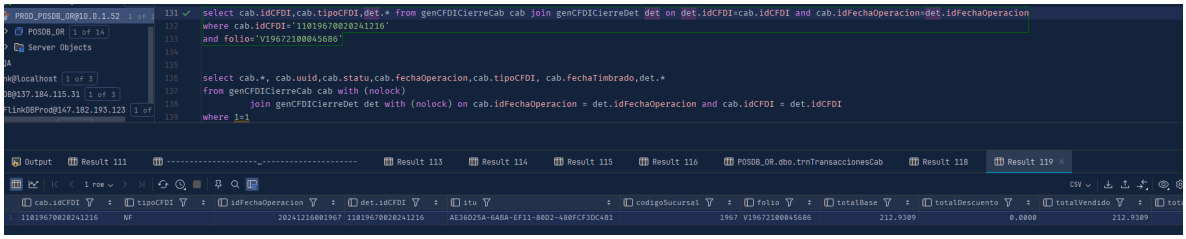
```
[11:56:40] ERROR> Excepción en la ejecución del thread: Object reference not set to an instance of an object.
[11:56:40] ERROR> IDCFDI='11023220020241218' folio='V23222200063656' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:40] ERROR> IDCFDI='11016260020241216' folio='V16262100040012' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:40] DEBUG> 11016260020241216 time to load 1356 isOKtoProcess=False
[11:56:40] ERROR> Excepción en la ejecución del thread: Object reference not set to an instance of an object.
```

Lo que demuestra lo que se había comentado.

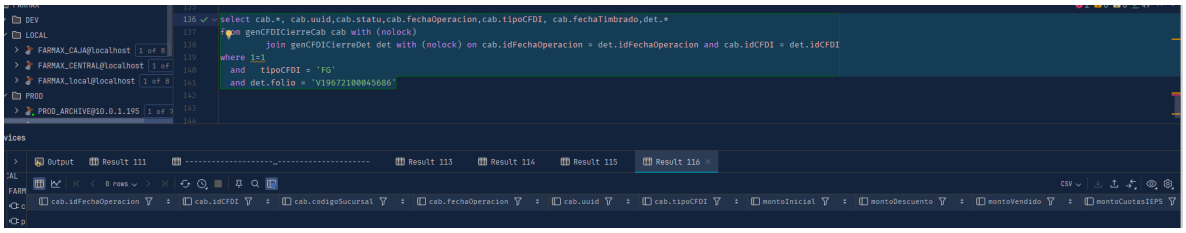
Ahora si revisamos las notas que están ocasionando el problema en esta ejecución por poner algunos ejemplos son:

[11:56:40] ERROR> IDCFDI='11019670020241216' folio='V19672100045686' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:40] ERROR> IDCFDI='11023220020241218' folio='V23222200063656' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:40] ERROR> IDCFDI='11016260020241216' folio='V16262100040012' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:40] ERROR> IDCFDI='11003130020241217' folio='V03132000039532' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:40] ERROR> IDCFDI='11026820020241216' folio='V26822200030133' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:41] ERROR> IDCFDI='11032960020241217' folio='V32962200001536' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:41] ERROR> IDCFDI='11023230020241218' folio='V23232100153989' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:42] ERROR> IDCFDI='11023230020241218' folio='V23232100153990' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:42] ERROR> IDCFDI='11032940020241218' folio='V32942200000958' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:42] ERROR> IDCFDI='11026360020241216' folio='V26362200050839' = '' isOKtoProcess=False TIPOCFDI='NF'
[11:56:42] ERROR> IDCFDI='11015800020241216' folio='V15802200036440' = '' isOKtoProcess=False TIPOCFDI='NF'

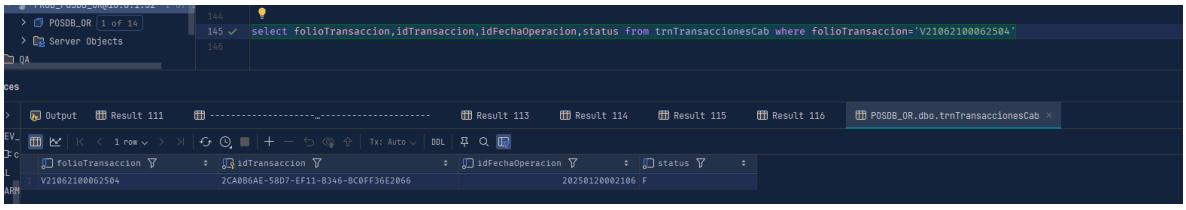
Si revisamos como esta en bd alguna de ellas, vemos que efectivamente la nota de crédito existe, es de tipo NF y que la venta asociada '**V19672100045686**' está asociada a dicha nota



Por otro lado vemos que no existe ninguna factura global que contenga en su detalle a la venta '**V19672100045686**'



Y que la venta '**V19672100045686**' esta facturada



Eso por supuesto es lo que está haciendo que la condición ***reluuid != null && !string.IsNullOrEmpty(reluuid.uuid)*** *No se cumpla y que* ***isOKToProcess*** *sea* ***false*** *y que por tanto la respuesta del método* ***GetCFDIRequest*** *de como resultado null*

```
private static InvoiceRequestGlobal GetCFDIRequest(DatabaseTools tools, GENCFDICIERRECAB item)
{
    bool isOKtoProcess = true;
    // set the description according to the CFDI type
    foreach (var scfdi :InvoiceCFDI in sales){...}
    // calculate header totals from rounded values
    if (Logger.IsDebug){...}
    if (isOKtoProcess)
    {
        return request;
    }
    return null;
```

Produciendo una excepción al tratar de escribir en log

```
InvoiceRequestGlobal cfdi = GetCFDIRequest(tools, cab);
s1.Stop();
Logger.AddDebugSubModule(SubModule, message: $"{cfdi.IdCFDI} request millis {s1.ElapsedMilliseconds}");
```

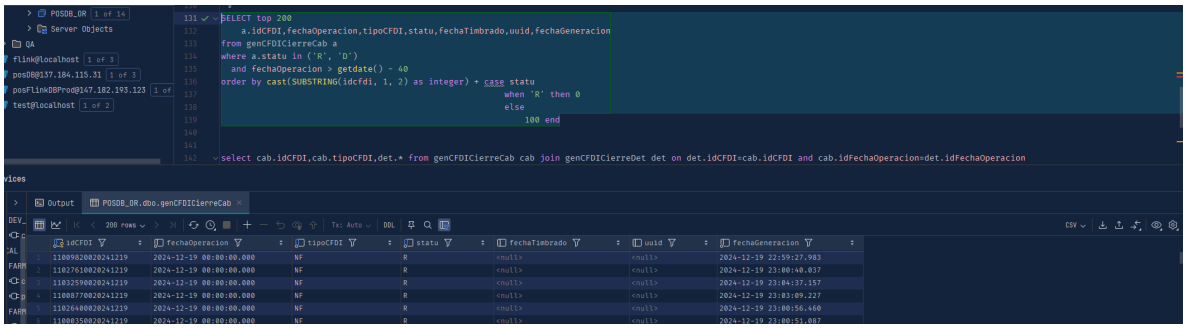
Sin embargo **aún no se contesta** la pregunta del porqué las notas de crédito tipo **ND** se “dejaron” de timbrar, y para eso debemos analizar la manera en que se seleccionan los registros a timbrar en el proceso y esto se hace mediante

```
private bool SearchPendingInvoice(out List<GENCFDICIERRECAB> listCFDIs, DatabaseTools tools)
{
    ProcessResult rc = new ProcessResult();
    listCFDIs = new List<GENCFDICIERRECAB>();

    try
    {
        // order by a priority where received FGs are attempted first, deferred CFDIs are given lower priority
        SqlReportQuery queryObj = new SqlReportQuery();
        string QUERY = $"SELECT top {BATCH_SIZE} a.idFechaOperacion, a.idCFDI, a.codigoSucursal, a.fechaOperacion, a.uuid, a.tipoCFDI, a.montoInicial, a.montoDescuento, a.montoVendido
        queryObj.Query.Statement = QUERY;
        queryObj.MappingObjects.Add(GENCFDICIERRECAB.DatabaseMappingObject);
        listCFDIs = tools.ExecuteQueryObjects<GENCFDICIERRECAB>(queryObj).ToList();
```

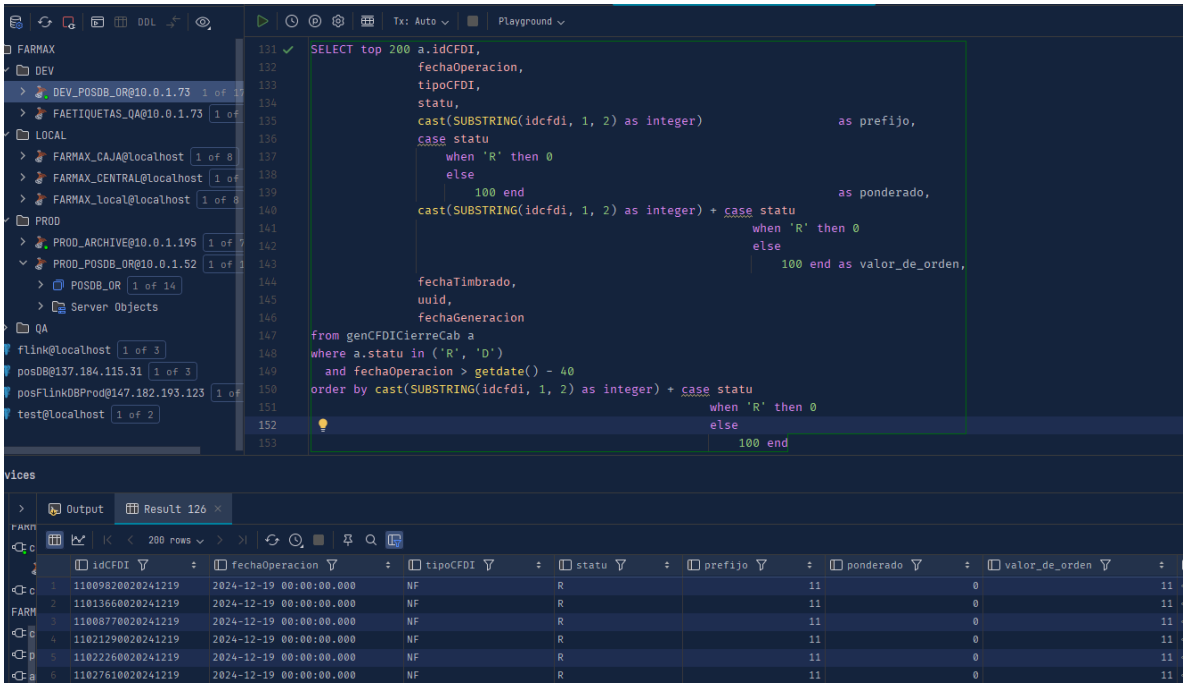
el query ejecutado, que es el siguiente:

```
SELECT top 200
   a.idCFDI,fechaOperacion,tipoCFDI,statu,fechaTimbrado,uuid,fechaGeneracion
from genCFDICierreCab a
where a.statu in ('R', 'D')
 and fechaOperacion > getdate() - 40
order by cast(SUBSTRING(idcfdi, 1, 2) as integer) + case statu
                                                      when 'R' then 0
                                                      else
                                                          100 end
```

Por lo que se observa hay un tipo especial de ordenamiento por los primeros dos dígitos del id de cfdi y por el statu, así que para entender mejor desmantelemos este ordenamiento en sus componentes para saber qué es lo que ocurre ya que si lo ejecutamos tal cual, prácticamente aparecen al inicio las facturas globales FG y después las NF y dejando fuera las ND ya que solo se consideran hasta 200 registros a procesar.



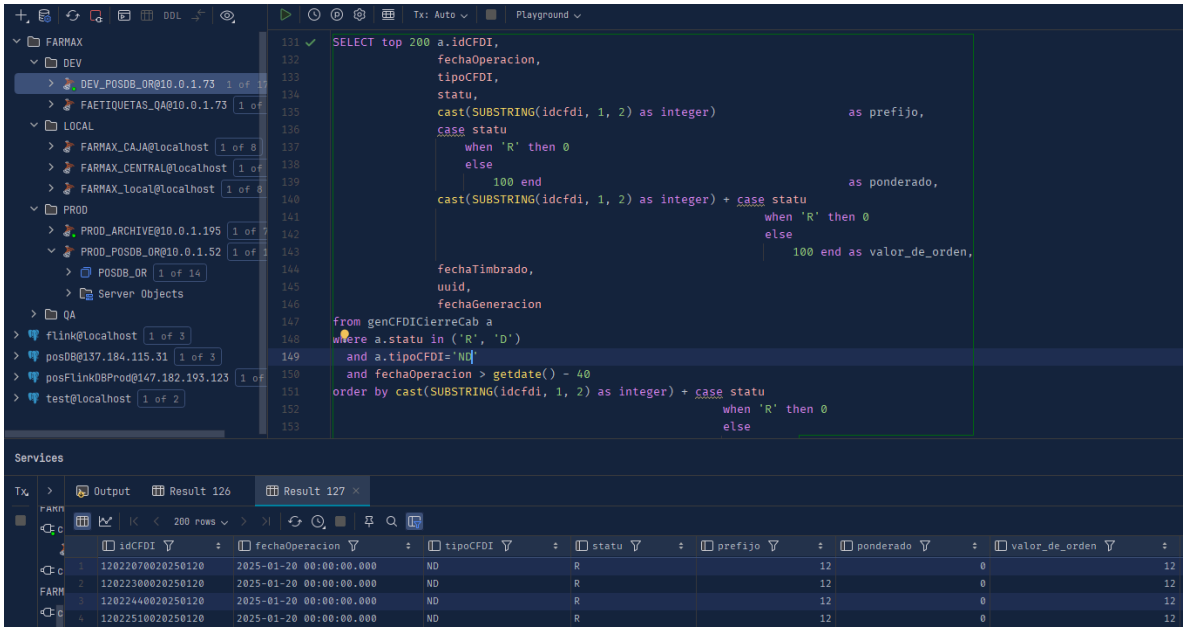
Modificamos un poco el query para que nos arroje el valor por el que ordena así como los componentes que integran este ordenamiento, se observa que el prefijo para las notas de crédito es **11**, dado que su status es **R** el ponderado es **0; entonces,** el valor para ordenar es **11**



```
SELECT top 200 a.idCFDI,
               fechaOperacion,
               tipoCFDI,
               statu,
               cast(SUBSTRING(idcfdi, 1, 2) as integer)                    as prefijo,
               case statu
                   when 'R' then 0
                   else
                       100 end                                             as ponderado,
               cast(SUBSTRING(idcfdi, 1, 2) as integer) + case statu
                                                              when 'R' then 0
                                                              else
                                                                  100 end as valor_de_orden,
               fechaTimbrado,
               uuid,
               fechaGeneracion
from genCFDICierreCab a
where a.statu in ('R', 'D')
  and fechaOperacion > getdate() - 40
order by cast(SUBSTRING(idcfdi, 1, 2) as integer) + case statu
                                                        when 'R' then 0
                                                        else
                                                            100 end
```

idCFDI	fechaOperacion	tipoCFDI	statu	prefijo	ponderado	valor_de_orden
1	11009820020241219	2024-12-19 00:00:00.000	NF	R	11	0	11
2	11013660020241219	2024-12-19 00:00:00.000	NF	R	11	0	11
3	11008770020241219	2024-12-19 00:00:00.000	NF	R	11	0	11
4	11021290020241219	2024-12-19 00:00:00.000	NF	R	11	0	11
5	11022260020241219	2024-12-19 00:00:00.000	NF	R	11	0	11
6	11027610020241219	2024-12-19 00:00:00.000	NF	R	11	0	11

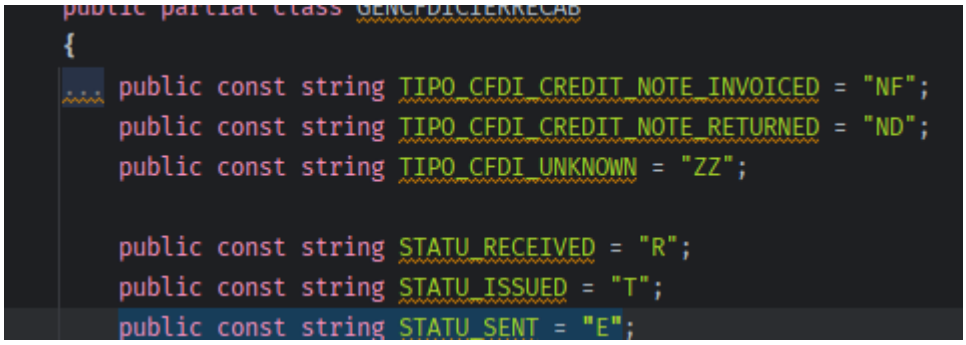
Ajustemos la consulta para que obliguemos a que los resultados sean de notas de crédito tipo **ND**, y se puede observar que para este caso el valor por el cual se ordena es **12**

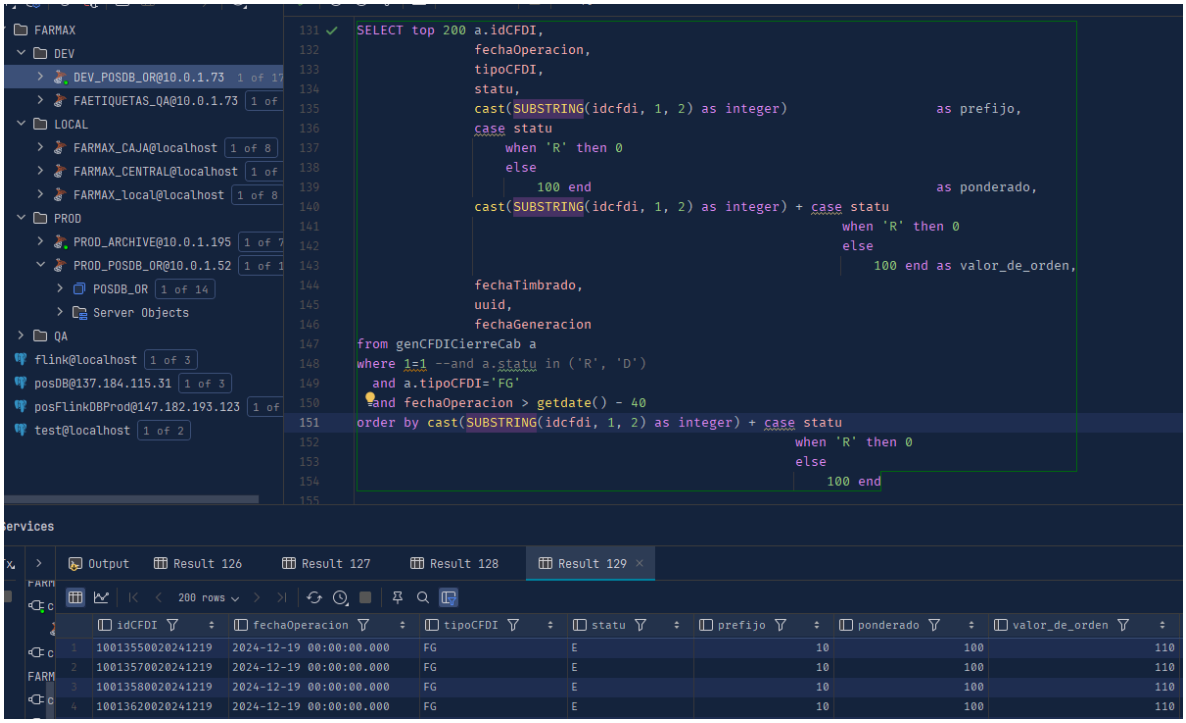


```
SELECT top 200 a.idCFDI,
               fechaOperacion,
               tipoCFDI,
               statu,
               cast(SUBSTRING(idcfdi, 1, 2) as integer)                    as prefijo,
               case statu
                   when 'R' then 0
                   else
                       100 end                                             as ponderado,
               cast(SUBSTRING(idcfdi, 1, 2) as integer) + case statu
                                                              when 'R' then 0
                                                              else
                                                                  100 end as valor_de_orden,
               fechaTimbrado,
               uuid,
               fechaGeneracion
from genCFDICierreCab a
where a.statu in ('R', 'D')
  and a.tipoCFDI='ND'
  and fechaOperacion > getdate() - 40
order by cast(SUBSTRING(idcfdi, 1, 2) as integer) + case statu
                                                        when 'R' then 0
                                                        else
```

idCFDI	fechaOperacion	tipoCFDI	statu	prefijo	ponderado	valor_de_orden
1	12022070020250120	2025-01-20 00:00:00.000	ND	R	12	0	12
2	12022300020250120	2025-01-20 00:00:00.000	ND	R	12	0	12
3	12022440020250120	2025-01-20 00:00:00.000	ND	R	12	0	12
4	12022510020250120	2025-01-20 00:00:00.000	ND	R	12	0	12

Para el caso de las facturas globales, para cuando se está haciendo esta consulta, no se tienen facturas globales pendientes de timbrar, es decir con estatus **R,** aun así podemos hacer una consulta para traer datos de facturas globales y ver cómo está funcionando el ordenamiento en ellas. Y como se puede observar el prefijo de la facturas globales es 10, el estatus es **E** por lo que su ponderado es 100 y su valor de ordenamiento es **110**, lo cual hace sentido porque es una factura con estatus enviada **E**, sin embargo, cuando la factura global no está timbrada tiene estatus **R,** por lo que su valor de ordenamiento cuando una factura global no está timbrada debe ser de **10 + 0 =10**





Conclusión

Después del análisis anterior podemos tener la causa raíz del porqué las notas de crédito ND se dejaron de timbrar y es un conjunto de factores:

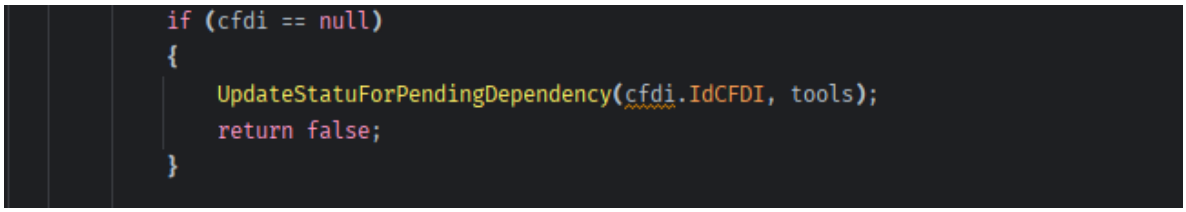
- Primero la selección de registros a ser timbrados se toman de acuerdo al valor del ordenamiento y un subconjunto de 200 de ellos, quedando la prioridad 10 **FG** (factura global), 11 notas de crédito (**NF**) y 12 notas de crédito (**ND**), de tal manera que si se acumulan más de 200 registros entre facturas globales y notas de crédito **NF**, se dejan fuera las notas de crédito **ND**
- Ahora por lo que se observa las facturas globales no han dejado de timbrarse por lo que estas van quedando fuera conforme se van procesando; sin embargo, hay notas de crédito **NF** que entran en el patrón que del error descrito (error con anterioridad) donde existe la nota de crédito **NF** pero no hay registro de sus ventas en alguna factura global por lo que se van acumulando, al menos más de 200 y esto cada día por lo que las notas de crédito **ND** quedan siempre fuera de la selección y por lo tanto sin timbrar.

Solución

Hay dos posibilidades, la primera que el código donde se intenta imprimir al log se comente

O validar que el objeto no venga null

Para dejar que el flujo:



Continúe.