UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Trabajo Almacenes de Datos

Integración de datos (ETL) de un almacén de UCI Sanitaria

Realizado por
De Pablo Diego y Soriano Juan

Profesor encargado:
Luque Baena Rafael Marcos

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, DICIEMBRE de 2024

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

ESTUDIANTES DE INGENIERÍA BIOINFORMÁTICA

Integración de datos (ETL) de un almacén de UCI Sanitaria

Almacenes de Datos

Realizado por
De Pablo Diego y Soriano Juan

Profesor encargado:
Luque Baena Rafael Marcos

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, DICIEMBRE DE 2024

Contents

1 Introducción **3**

2 Objetivos **3**

3 Modificación del almacén de datos **3**
3.1 Ingreso a la UCI 4
3.2 Tiempo de Alta 5
3.3 Paciente 5
3.4 Supresión de Respiratory Charting 5
3.5 Almacenamiento del identificador del paciente de la base de datos original 5
3.6 Relaciones entre Tablas 5

4 ETL del almacén de datos NorthwindDW **7**
4.1 Dificultades en Northwind 7

5 ETL del almacén de datos de pacientes con patologías respiratorias **8**

6 ETL del almacén de datos de pacientes con patologías respiratorias **8**
6.1 BorrarAlmacenUCI 10
6.2 Carga de Género y Etnia 11
6.3 Carga de Región 12
6.4 Carga Tiempo 12
6.5 Carga Priority 13
6.6 Carga Tipo 13
6.7 Carga RouteAdmin 13
6.8 Carga Hospital 14
6.9 Carga Paciente 15
6.10 Carga IngresoUCI 18
6.11 Carga ApacheApsVar 20
6.12 Carga AdmissionDX 21
6.13 Carga RespiratoryCare 22
6.14 Carga Diagnosis 22
6.15 Carga UCI Diagnosis 24
6.16 Carga Alergia 24
6.17 Carga UCI Alergia 25
6.18 Carga Medicacion 25
6.19 Carga UCI Medicacion 26

7 Dificultades encontradas **26**

8 Conclusiones **27**

9 Conclusión **27**

10 Github y conjunto de instrucciones para su correcto despliegue en SQL Server. **27**

1 Introducción

Este documento constituye una continuación del trabajo previo, donde se desarrolló el diseño conceptual y lógico de un almacén de datos basado en la información proporcionada por la *Base de Datos de Investigación Colaborativa eICU* [1]. En ese proyecto, el enfoque principal fue la selección y análisis de información relativa a **pacientes con patologías respiratorias**, determinando las tablas y columnas más relevantes para permitir un análisis exhaustivo de esta población específica.

En esta nueva fase, se procederá a la implementación del proceso de **Extracción, Transformación y Carga** (ETL, por sus siglas en inglés). Este proceso es fundamental para la integración de datos en cualquier almacén de datos, ya que permite extraer datos de múltiples fuentes, transformarlos según las necesidades del modelo, y finalmente cargarlos en el sistema de almacenamiento. El proceso ETL es clave para garantizar la calidad, consistencia e integridad de los datos, factores esenciales para que el análisis posterior sea preciso y confiable.

El éxito de este proceso asegura que las tablas del almacén de datos estén adecuadamente pobladas con información precisa, sentando las bases para un **análisis de datos** eficiente. Este proceso facilita la realización de consultas complejas o la integración de herramientas como *Reporting Services*, que permiten la visualización clara y sencilla de la información más relevante, apoyando la toma de decisiones clínicas fundamentadas.

Por lo tanto, este documento también incluirá un tutorial detallado del proceso de carga para las tablas del almacén de datos personalizado. Además, se presentará un análisis de las dificultades encontradas durante la implementación y las estrategias empleadas para superarlas.

2 Objetivos

El principal objetivo de este informe es documentar de manera detallada la ejecución del proceso ETL en dos contextos diferentes:

- El almacén de datos *NorthwindDW*, utilizado como referencia durante las sesiones prácticas, cuya carga será replicada siguiendo los procedimientos previamente establecidos.

- El almacén de datos del *eICU*, adaptado específicamente para el análisis de **pacientes con patologías respiratorias**, que será implementado utilizando el modelo lógico desarrollado en la fase anterior del proyecto.

A lo largo del documento se mostrará cómo se ha llevado a cabo la integración de datos en ambos almacenes, resaltando los desafíos enfrentados y las soluciones aplicadas, con el fin de proporcionar una guía clara y replicable del proceso.

3 Modificación del almacén de datos

Durante la implementación del almacén de datos se realizaron diversas modificaciones en su estructura original, con el fin de adaptarlo lo mejor posible siguiendo

las mejores prácticas para crear un buen almacén de datos. A continuación, se detallan los principales cambios efectuados y la justificación detrás de cada uno de ellos (puede ver también la figura 1).

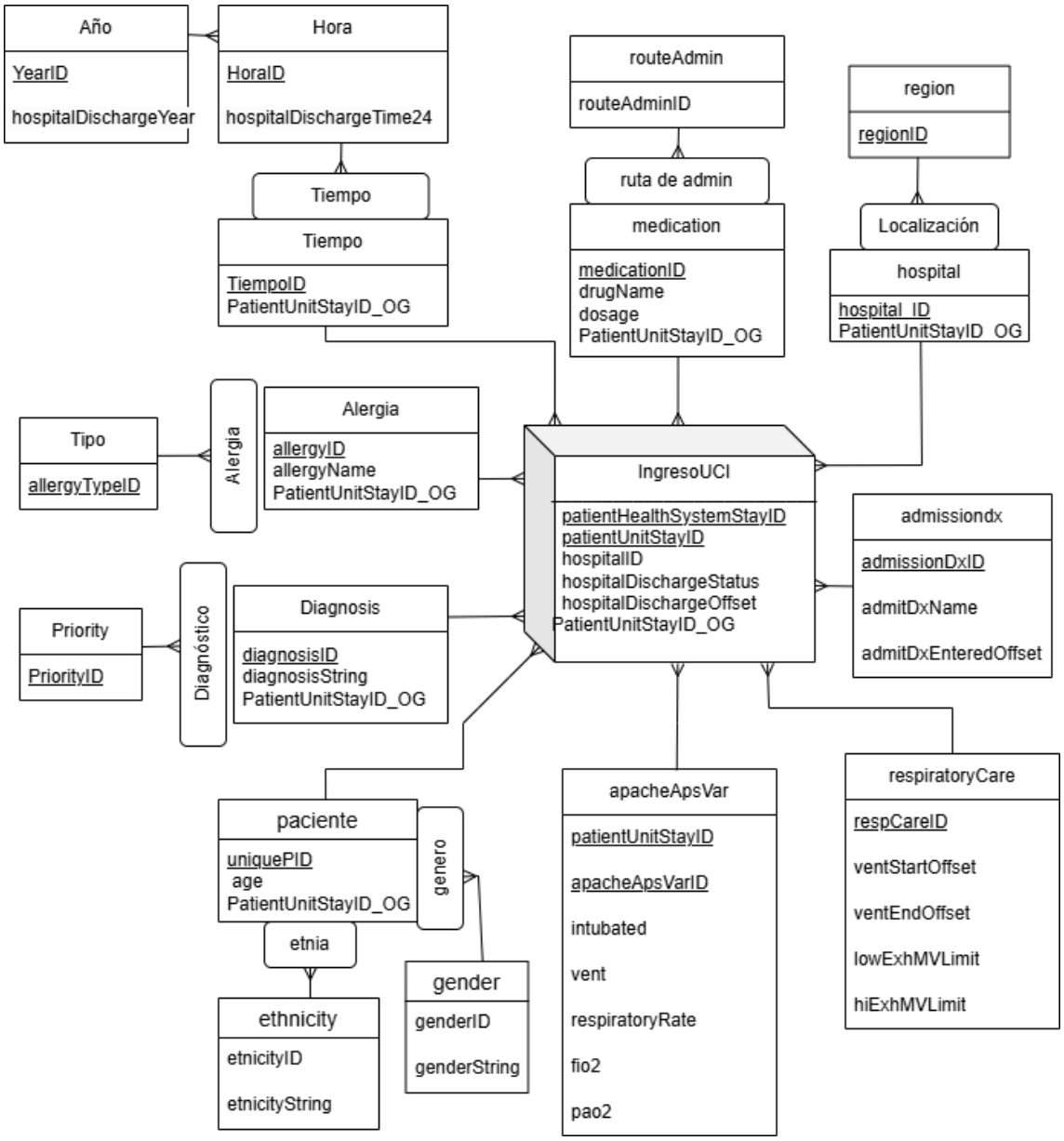

Figure 1: Diagrama actualizado (Correcciones implementadas)

3.1 Ingreso a la UCI

El mayor cambio que hubo es que el hecho Ingreso a la UCI fue separado de la tabla paciente, Además se incorporó el atributo `hospitalDischargeOffset`, que indica la duración de la estancia del paciente en la UCI, medida en minutos desde el ingreso hasta el alta hospitalaria. Este ajuste fue sugerido para permitir el análisis del tiempo de hospitalización de cada paciente, un factor relevante en los estudios de recuperación y tratamiento de enfermedades respiratorias.

3.2 Tiempo de Alta

Para estudiar el tiempo en este proyecto se decidió almacenar únicamente los atributos `hospitalDischargeTime24` y `hospitalDischargeYear`, ya que la base de datos no contiene un campo `hospitalAdmitYear`, como se había sugerido originalmente. Esta decisión se tomó con base en la disponibilidad de datos y la coherencia con el diseño del modelo lógico.

3.3 Paciente

La tabla **Paciente**, cuyo identificador único (`PK`) es el campo `uniquePID`. Además de este identificador, se incluyeron atributos relevantes como la `edad`. También se estableció una jerarquía paralela para **género** y **etnia**, lo que permitió organizar estos atributos de forma más estructurada y lógica.

3.4 Supresión de Respiratory Charting

La tabla **Respiratory Charting**, fue eliminada al trabajar más detenidamente en ella se observó múltiples inconsistencias como es la baja cantidad de datos en comparación a otros datos, cierto parentesco para algunas métricas que ya se calculaban en *ApacheApsVar* y que dependiendo del paciente se le hacía una métrica especifica, lo cual terminaba siendo rellenado con cierta cantidad de errores. En búsqueda de mantener un almacén más claro y simple se elimina esta tabla del DDL.

3.5 Almacenamiento del identificador del paciente de la base de datos original

El proceso ETL es complejo y puede generar dependencias para mantener la coherencia de las relaciones entre las tablas. En el caso de los datos de eICU, existen múltiples tablas con relaciones de tipo M:N con el hecho principal, lo que requiere la creación de tablas intermedias en el almacén de datos. Para establecer estas relaciones, es necesario incluir un atributo que vincule las tablas de forma adecuada. Por esta razón, se creó el atributo *PatientUnitStayID_OG*, que almacena el identificador del paciente original de la base de datos eICU. Este identificador es esencial, ya que al migrar los datos al nuevo almacén, se utilizará para vincular las nuevas claves autogeneradas.

Un ejemplo de este proceso es la tabla de alergias, que, además de almacenar el nombre de la alergia y si está relacionada con un fármaco, incluye el identificador de los ingresos asociados a dicha alergia. Esto es necesario porque, en este proyecto, las claves originales fueron reemplazadas por valores autogenerados.

3.6 Relaciones entre Tablas

A continuación se describen las relaciones establecidas entre las tablas del almacén de datos, modificadas para garantizar un diseño coherente y funcional:

1. **Paciente**
   Relación: 1-n
   Cada paciente puede tener múltiples ingresos a la UCI, lo que refleja que un

mismo paciente puede ser readmitido en distintos momentos debido a recaídas o nuevas patologías.

2. **Diagnosis**
   Relación: n-m
   Un paciente puede tener múltiples diagnósticos asociados a un único ingreso, y el mismo diagnóstico puede repetirse en diferentes ingresos y entre distintos pacientes.

3. **Medicamentos**
   Relación: n-m
   Un ingreso puede estar asociado a la administración de varios medicamentos. Además, un medicamento puede ser utilizado en distintos ingresos de múltiples pacientes.

4. **Alergia**
   Relación: n-m
   Cada paciente puede tener múltiples alergias documentadas, las cuales son relevantes para su tratamiento durante cada ingreso a la UCI. Por lo tanto, una alergia puede estar relacionada con múltiples ingresos.

5. **RespiratoryCare**
   Relación: 1-n
   Similar a *RespiratoryCharting*, cada ingreso puede tener múltiples intervenciones de cuidado respiratorio, como la administración de oxígeno o ventilación mecánica, asociadas a un ingreso específico.

6. **apacheApsVar**
   Relación: 1-1
   Cada ingreso a la UCI tiene una única evaluación APS asociada. Dependiendo del diseño, esta relación puede ser de uno a uno (si se almacena como un único resumen por ingreso) o de uno a muchos (si se almacena como varios componentes individuales evaluados), se decidió que la primera forma otorga más información y mayor relación con el hecho.

7. **Admissiondx**
   Relación: 1-n
   Cada ingreso tiene un diagnóstico primario de admisión, aunque pueden existir diagnósticos más concretos que se documentan en otra tabla, como *Diagnosis* que contiene información más clara y estudiada.

8. **Hospital**
   Relación: 1-n
   Cada hospital puede tener múltiples ingresos a la UCI. Los ingresos se asocian exclusivamente a un hospital, dependiendo del centro de atención en el que se encuentre la unidad.

Los cambios en las relaciones entre tablas buscan optimizar el diseño del almacén de datos, adaptándolo a las necesidades específicas del análisis clínico de los pacientes con patologías respiratorias. Estas modificaciones garantizan la integridad de los datos y la flexibilidad para realizar análisis detallados en contextos hospitalarios.

4 ETL del almacén de datos NorthwindDW

En esta sección se muestra el proceso de la carga del almacén de datos *NorthwindDW*, siguiendo el modelo proporcionado en clase. El procedimiento se realizó de acuerdo a las pautas establecidas, y para confirmar la correcta ejecución de las cargas, se incluirán capturas de pantalla que muestran el estado final del proceso con todos los indicadores de éxito ("ticks" en verde). Además, se adjunta un archivo adicional en el reporte con más detalles sobre las ejecuciones y el resultado final.(Ver figura 2)

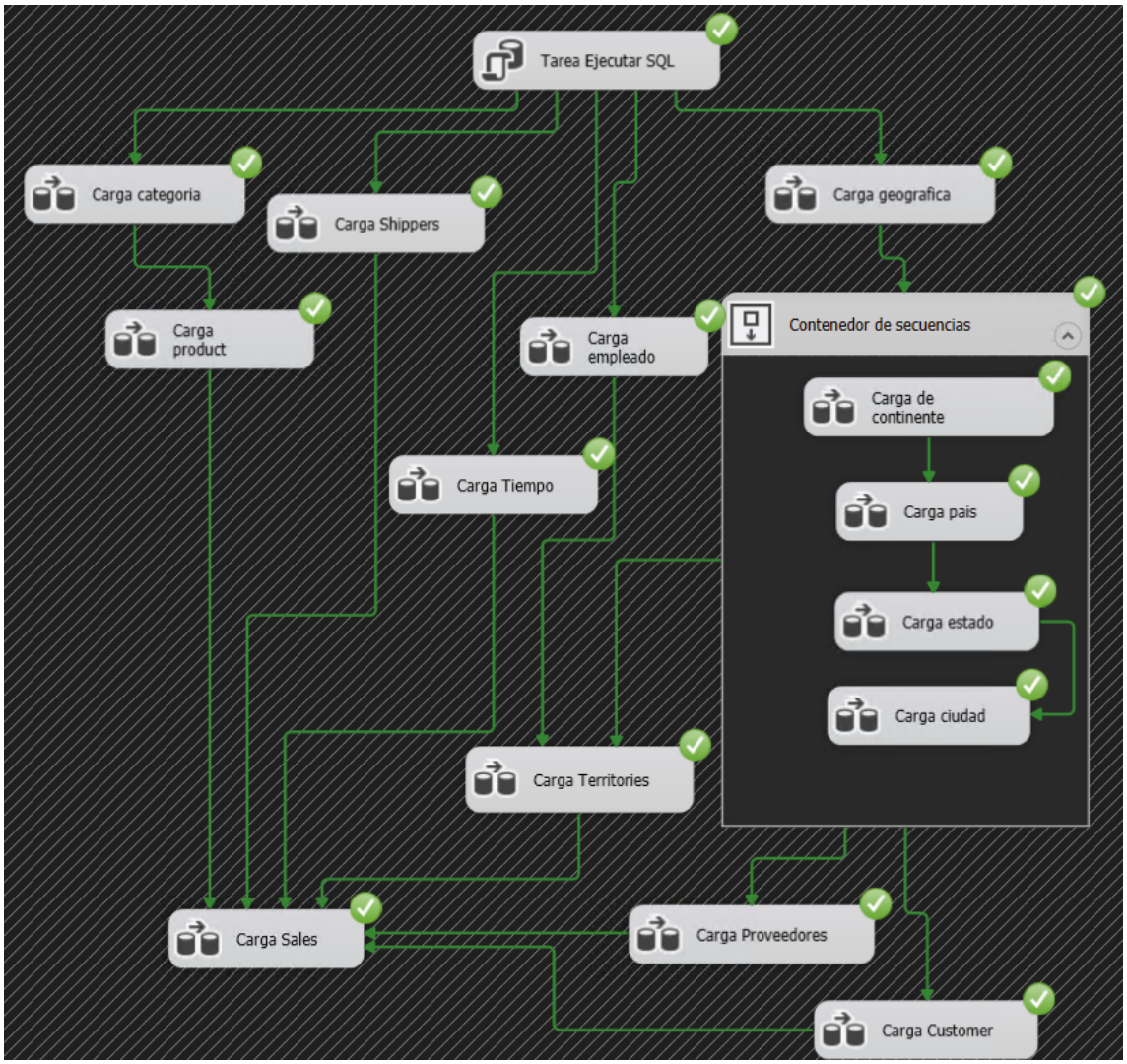

Figure 2: Flujo ETL Northwind completo

4.1 Dificultades en Northwind

Debido a que el archivo original, Time.xls, estaba en un formato antiguo, fue necesario transformarlo a Time.xlsx. Aunque se intentó especificar el origen como "Excel 1997", no fue posible reconocer la columna correctamente. Tras realizar la conversión al formato Excel 2007-2010, el sistema logró identificar sin problemas la columna Feuil.

Otra dificultad que enfrentamos fue durante el vaciado del almacén de datos, específicamente con las claves foráneas (FK). El problema surgía porque, en algunos casos, se eliminaban primero los registros asociados a la clave primaria (PK), lo que generaba conflictos con las claves foráneas dependientes. Para resolver este inconveniente, decidimos deshabilitar temporalmente las restricciones de las claves foráneas antes de realizar el vaciado del almacén y, una vez completado, las volvimos a habilitar.

– Desactivar las restricciones de clave foránea temporalmente

EXEC sp_MSForEachTable 'ALTER TABLE ? NOCHECK CONSTRAINT ALL';

– Reactivar las restricciones de clave foránea

EXEC sp_MSForEachTable 'ALTER TABLE ? WITH CHECK CHECK CONSTRAINT ALL';

5 ETL del almacén de datos de pacientes con patologías respiratorias

Antes de realizar el proceso ETL, es necesario dejar claras algunas decisiones tomadas en el llenado del **Data Warehouse (DW)**. Durante el desarrollo del proceso ETL. Aunque las claves foráneas de un hecho deben ser las claves primarias de las tablas relacionadas. Sin embargo, en nuestro caso, hemos optado por una solución que permite evitar trabajo extra al llenar las tablas intermedias, sin perder información. En lugar de utilizar tres claves foráneas diferentes, hemos creado una clave autogenerada como clave primaria (PK) que sea única para cada combinación de `TiempoID`, `HospitalID` y `UniquePID`. De esta forma, mantenemos la eficacia de acceder a las claves foráneas `TiempoID`, `HospitalID` y `UniquePID` desde otras tablas, pero sin la necesidad de escribir tres claves diferentes.

Con esta solución, se logró una estructura más flexible y escalable en la construcción del almacén de datos y facilitar futuras consultas y análisis.

6 ETL del almacén de datos de pacientes con patologías respiratorias

, específicamente en relación con el hecho. Según la teoría, las claves foráneas de un hecho deben ser las claves primarias de las tablas relacionadas. Sin embargo, en nuestro caso, hemos optado por una solución que permite evitar trabajo extra al llenar las tablas intermedias, sin perder información. En lugar de utilizar tres claves foráneas diferentes, hemos creado una clave autogenerada como clave primaria (PK) que sea única para cada combinación de `TiempoID`, `HospitalID` y `UniquePID`. De esta forma, mantenemos la eficacia de acceder a las claves foráneas `TiempoID`, `HospitalID` y `UniquePID` desde otras tablas, pero sin la necesidad de escribir tres claves diferentes.

Otra decisión importante fue almacenar ciertas claves antiguas en algunas tablas para facilitar la herencia de claves foráneas. Por ejemplo, en el hecho hemos almacenado la clave antigua del paciente, `patientUnitStayID_og`, con el objetivo de facilitar la herencia de la clave primaria del hecho `IngresoUCI`.

El resto de claves en las tablas serán autogeneradas. Un ejemplo de esto es el siguiente código para la tabla `Ethnicity`:

```
ALTER TABLE Ethnicity ADD ethnicityID INT IDENTITY(1,1) PRIMARY KEY;
```

Así mismo en este apartado se explicará más detenidamente todo el proceso de carga aplicado para este proyecto, partiendo desde el uso de *Microsoft SQL Server Management Studio* para restaurar las bases de datos, tanto la fuente de datos correspondiente a La base de datos de investigación colaborativa, *eICU Collaborative Research Database*, disponible en este enlace. Además del *DDL* creado específicamente para esta actividad, accesible en el repositorio de GitHub a través de este enlace. A partir de un enfoque para pacientes con patología respiratorias

Una vez restaurada las bases de datos y preparadas sus conexiones para el repositorio de *Visual Studio, SQL Server Integration Services Project*, se podrá empezar el proceso de ETL a nuestro nuevo Data Warehouse (DW), como se observa en la figura 3 al contar con gran cantidad de tablas en el modelo copo de nieve resultará bastantes flujos de datos en nuestro repositorio.

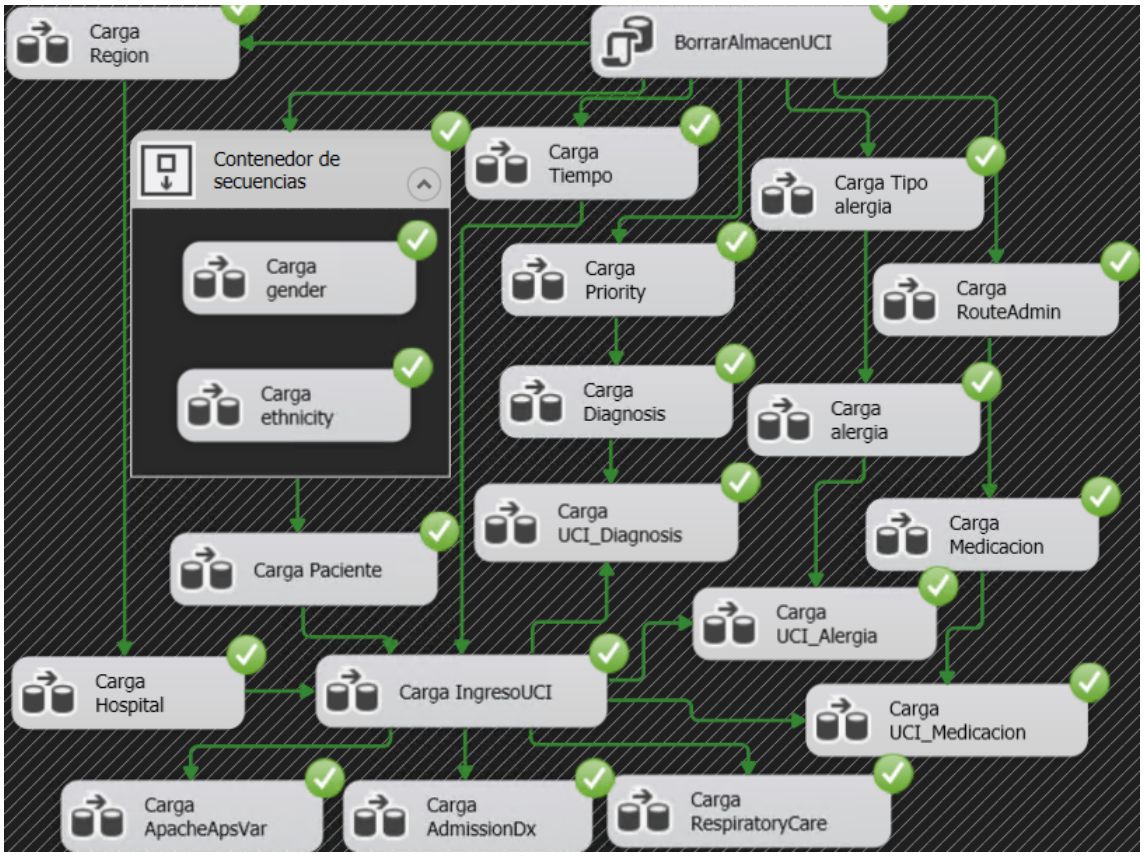

Figure 3: Flujo de control una vez acabado el ETL

A continuación la explicación tarea por tarea de cada actividad planteada en el flujo de control, es importante destacar que el trabajo sigue cierta secuencialidad en algunas secciones que dependen de fases anteriores, se íntentará destacar la paralelización de tareas y la tarea previa a cada actividad pero es importante seguir las flechas mostradas en la figura 3.

6.1 BorrarAlmacenUCI

La primera actividad por excelencia en cualquier proyecto de integración de servicios es **BorrarAlmacenUCI**, para evitar duplicar datos al crear la carga de datos, como se observa en la figura 4 se aplica una *Tarea Ejecutar SQL*, indicando el *SQL-Statment y IsQueryStoredProcedure*, para este punto se debe volver a *SQL Server Management* para crear un procedimiento almacenado (en la database, la carpeta *Programmability y StoredProcedures*), el cual se encargará de deshabilitar las restricciones de claves foráneas (útil para esta ocasión donde se desea eliminar todas las tablas para preparar el ETL y evitar errores por borrar tablas en un orden equivocado) se elimina los registros de cada tabla y por último habilita nuevamente las restricciones de claves foráneas, a través de este enlace podrá ver el código detalladamente.

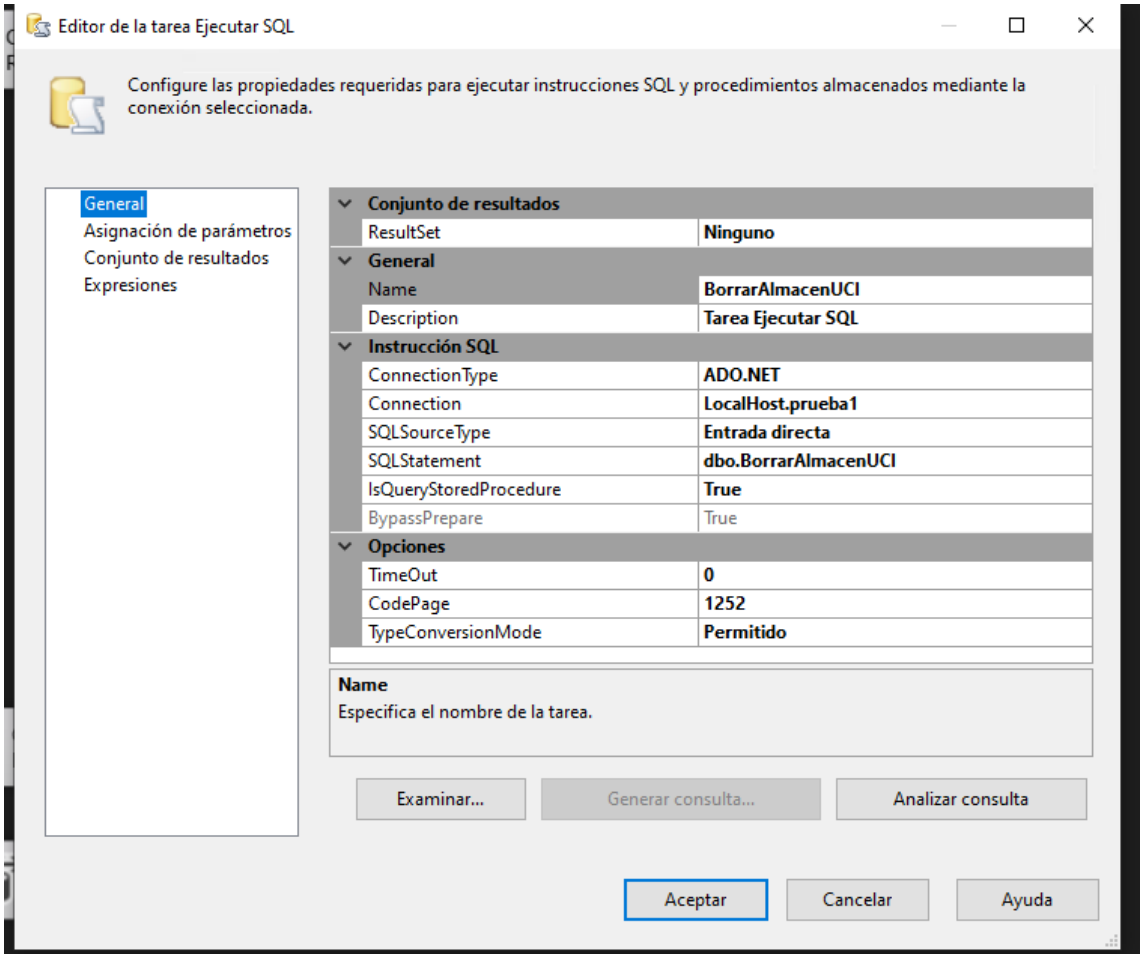

Figure 4: Ventana de información de BorrarAlmacenUCI, ya configurada

Una vez realizado el borrado de las cargas anteriores, se puede proceder con las tareas reales de carga del almacén. Es fundamental evitar la secuencialidad en el trabajo, por lo que los flujos de datos que puedan ejecutarse en paralelo deben iniciarse. Tras ejecutar el proceso *BorrarAlmacenUCI* (recomendable ver la figura 3 y a través de las flechas entender el orden de llenado de cargas), se inician inmediatamente los siguientes seis flujos de datos:

6.2 Carga de Género y Etnia

Debido a que género y etnia son jerarquícamente paralelas es decir que es deseable que se carguen a la vez y una vez ambas finalicen se usen para cargar la tabla paciente, para ello se usará un contenedor de secuencias.

Para la **carga de género** (ver Figura 5), se utilizó como origen la base de datos [**eICU Collaborative Research Database**]. En lugar de emplear directamente la tabla de Patient, se utilizó la siguiente consulta:

```
SELECT DISTINCT Gender FROM Patient
```

El objetivo de esta consulta fue almacenar únicamente los géneros distintos, incluyendo los valores **NULL** para evitar discriminar a pacientes cuyo género no fue registrado.

Adicionalmente, se realizó una conversión de datos, ya que en el almacén de destino el campo `genderString` está definido como una cadena Unicode (`DT_WSTR`) con una longitud de 25 caracteres. Finalmente, se almacenaron los géneros distintos en el destino, y la clave `genderID` fue autogenerada en la nueva base de datos.

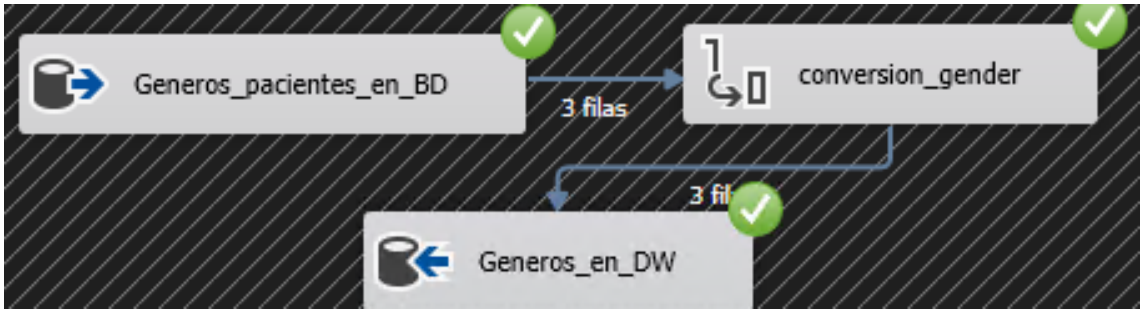

Figure 5: Carga géneros

En la **carga de etnia**, se aplicó la misma lógica utilizada en la carga de géneros. Se llevó a cabo una conversión de los datos para ajustar el campo `Ethnicity` del origen al campo `ethnicityString` en el destino. Tras procesar los valores, estos fueron almacenados correctamente en el nuevo almacén de datos.

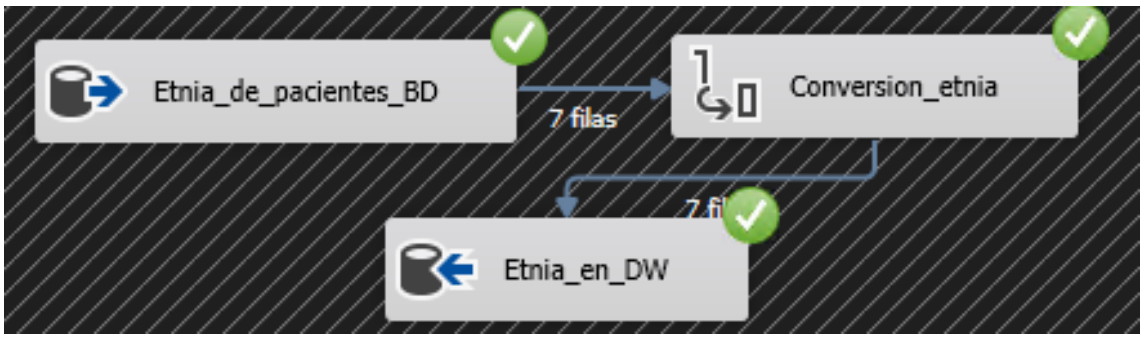

Figure 6: Carga etnia

6.3 Carga de Región

En el caso de **Carga de Región**, el flujo de datos es muy sencillo, como se observa en la figura 7. Este proceso consta de dos herramientas principales: un origen **ADO NET**, que accede a la base de datos `eICU` para consultar las distintas regiones presentes en la tabla `hospital`, y un destino **ADO NET**, vinculado a la tabla `Region` del Data Warehouse. En este flujo, el atributo `regionString` se utiliza para almacenar el nombre de la región, mientras que el atributo `regionID` es una clave autogenerada, por lo que no es necesario rellenarla manualmente.

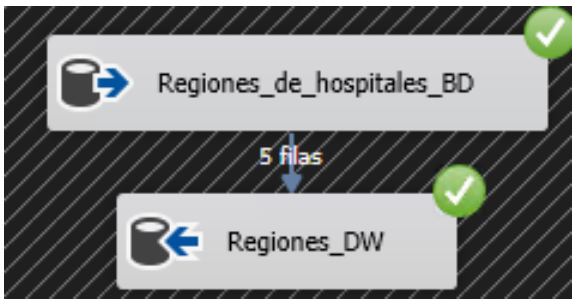

Figure 7: Carga de las distintas regiones para los hospitales de la base de datos elCU

En este caso, los datos originales de la base de datos `eICU` contienen únicamente 5 filas correspondientes a `NULL`, `Midwest`, `Northeast`, `South` y `West`, que son las cinco posibles regiones asignadas a los hospitales. Durante la carga de las distintas tablas, es posible observar un número diferente de filas, lo cual no debe interpretarse como un error. Esto se debe a que los 186 hospitales de la base de datos están conectados a la tabla `Region`, donde cada hospital se vincula con su respectiva región.

6.4 Carga Tiempo

Para la **carga de la tabla tiempo 8**, únicamente se trasladaron las tuplas de la tabla `Paciente` de la base de datos del eICU,específicamente los campos:

`HospitalDischargeTime24` y `HospitalDischargeYear`. El campo `TiempoID`, que se encuentra en el destino, fue autogenerado automáticamente al realizar el almacenamiento de los datos.

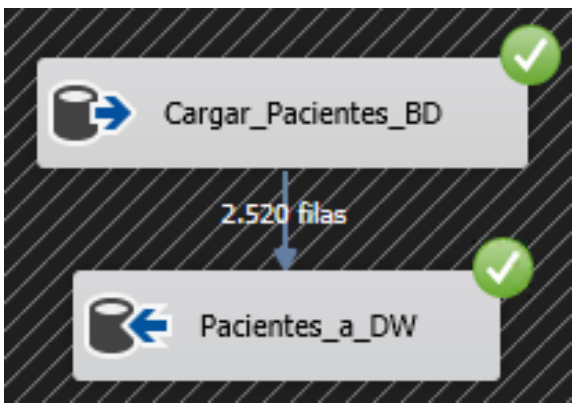

Figure 8: Carga Tiempo

6.5 Carga Priority

Para la **carga de prioridad** se siguió exactamente el mismo procedimiento utilizado para género y etnia. Se seleccionaron únicamente las distintas prioridades asociadas a los diagnósticos mediante la siguiente consulta:

```
SELECT DISTINCT DiagnosisPriority FROM Diagnosis
```

Posteriormente, los valores obtenidos se insertaron en el **Data Warehouse (DW)**.

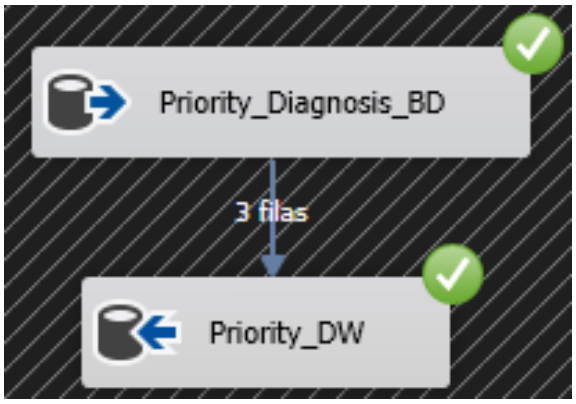

Figure 9: Carga Priority

6.6 Carga Tipo

Al igual que en anteriores cargas esta viene a ser exactamente el mismo procedimiento obteniendo los datos de ElCU a través de una consulta que otorgue las 3 prioridades posibles y estas se guardan en la tabla tipo.

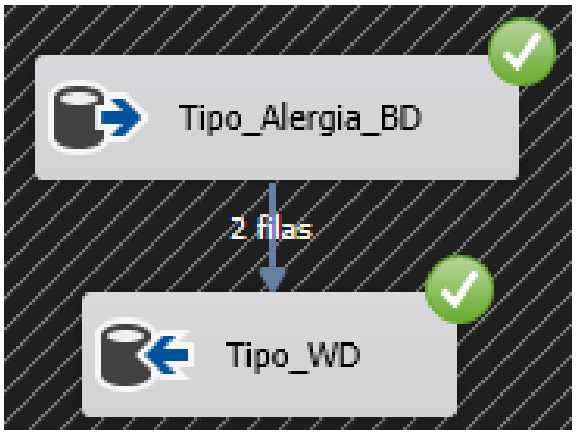

Figure 10: Carga Tipo

6.7 Carga RouteAdmin

Este es el último proceso que se realiza paralelamente despúes del uso de la tarea Ejecutar SQL: BorrarAlmacenUCI, como viene siendo sus contrapartes es la carga de una dimensión en este caso para la medicación.

En el caso de esta carga la diferencia a destacar es que se uso la herramienta columna derivada para poner en mayúsculas el texto de RouteAdmin, esto sé hizo porque en la base de datos ElCU existen variaciones del mismo termino como puede ser el caso de oral, que indica que un medicamento se toma por esta vía, al existir las variaciones como pueden ser Oral, ORAL y oral, la base de datos estaría identificando como diferente estas 3 palabras, cuando se refieren al mismo método.

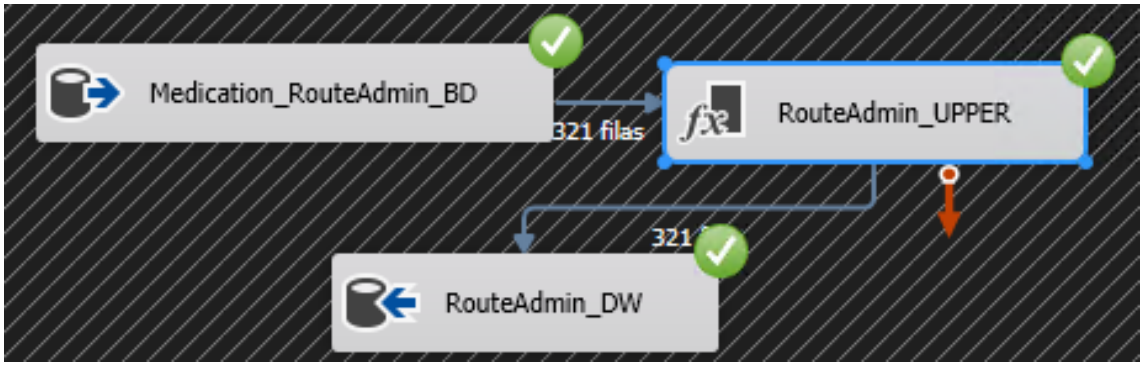

Figure 11: Carga RouteAdmin

6.8 Carga Hospital

Antes de que se llegue a compilar la carga hospital es necesario que la carga de región haya terminado (Observar la sección 6.3), esto se debe a que esta tabla cargara datos del propio almacén buscando conectar los hospitales con las regiones ya almacenadas.

Para la **carga de hospital**, se utilizó como origen la tabla `Hospital` de la base de datos **eICU**. Posteriormente, se realizó una conversión del atributo `Region` de la tabla `Hospital`, transformándolo a una cadena (`DT_STR`) con una longitud de 64 caracteres.

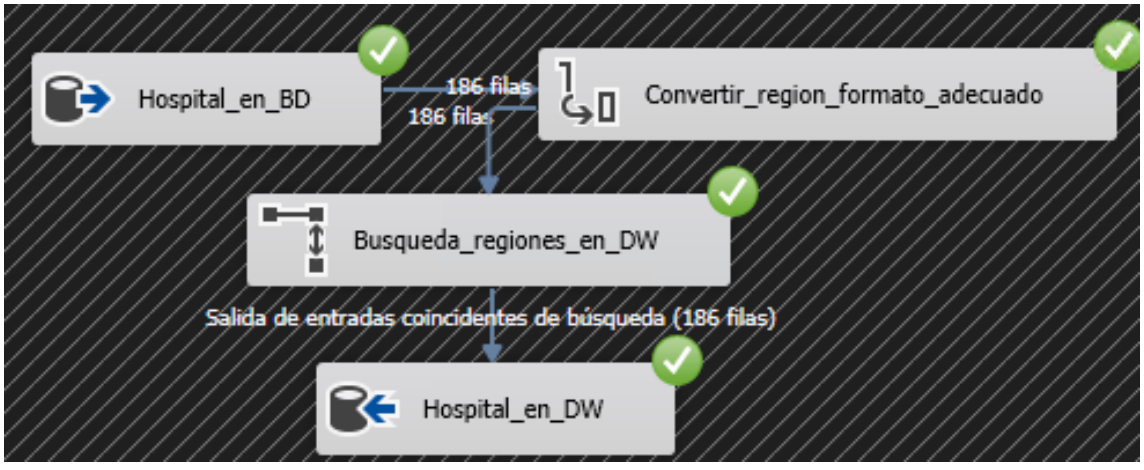

Figure 12: Carga Hospital

Esta transformación fue necesaria para realizar un *join* mediante un elemento *lookup* de **SQL Integration Services**, cuyo objetivo fue buscar las tuplas que

coincidieran en región y, a partir de ahí, obtener el correspondiente `regionID`. (ver figura 13)

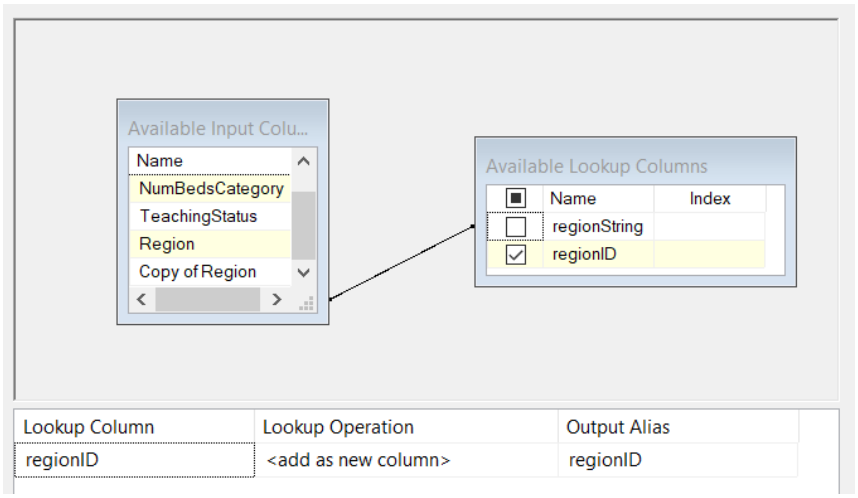

Figure 13: Busqueda_regiones_en_DW

Finalmente, se almacenaron los siguientes campos en el destino (ver figura14):

- `HospitalID` del **eICU**, renombrado como `hospitalID_og`, para permitir que `IngresoUCI` herede la nueva clave `HospitalID` generada en nuestro **DW**.
- `regionID`, como clave foránea heredada de la tabla `Region`.

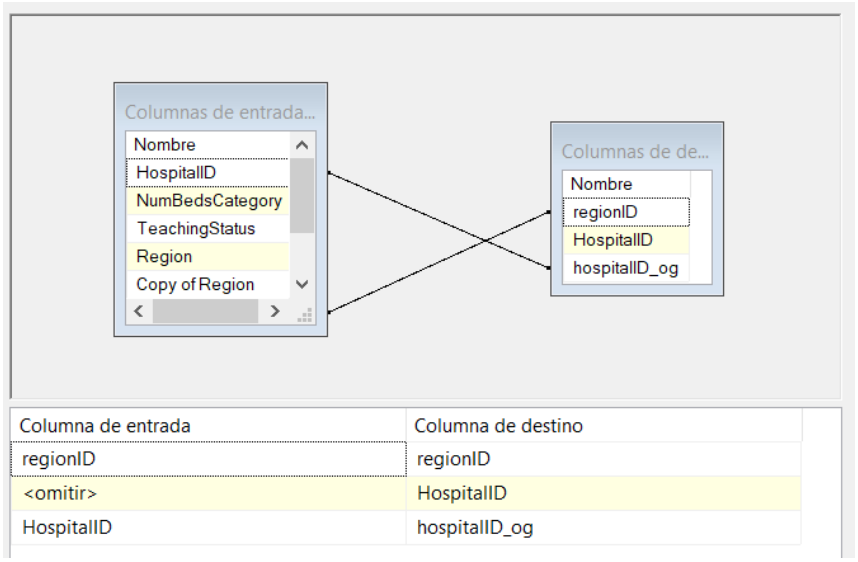

Figure 14: Hospital_en_DW

6.9 Carga Paciente

En el caso de la carga de paciente es necesario que se cargue el contenedor de secuencia que incluye la carga de gender y ethnicity (ver sección 6.2), el flujo de datos es el correspondiente a la figura 15

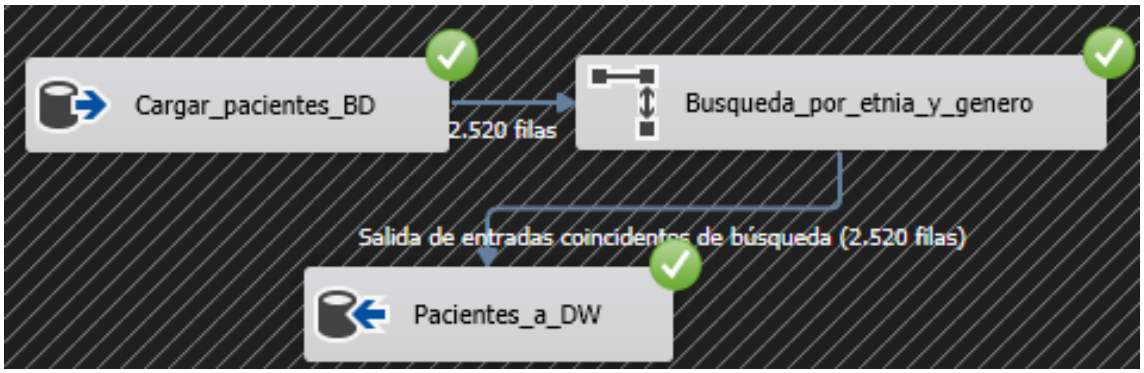

Figure 15: Carga Paciente

Para la **carga de paciente**, se utilizó como origen la tabla `Patient` de la base de datos **eICU**. En este proceso, se seleccionaron los atributos especificados en la Figura 16.

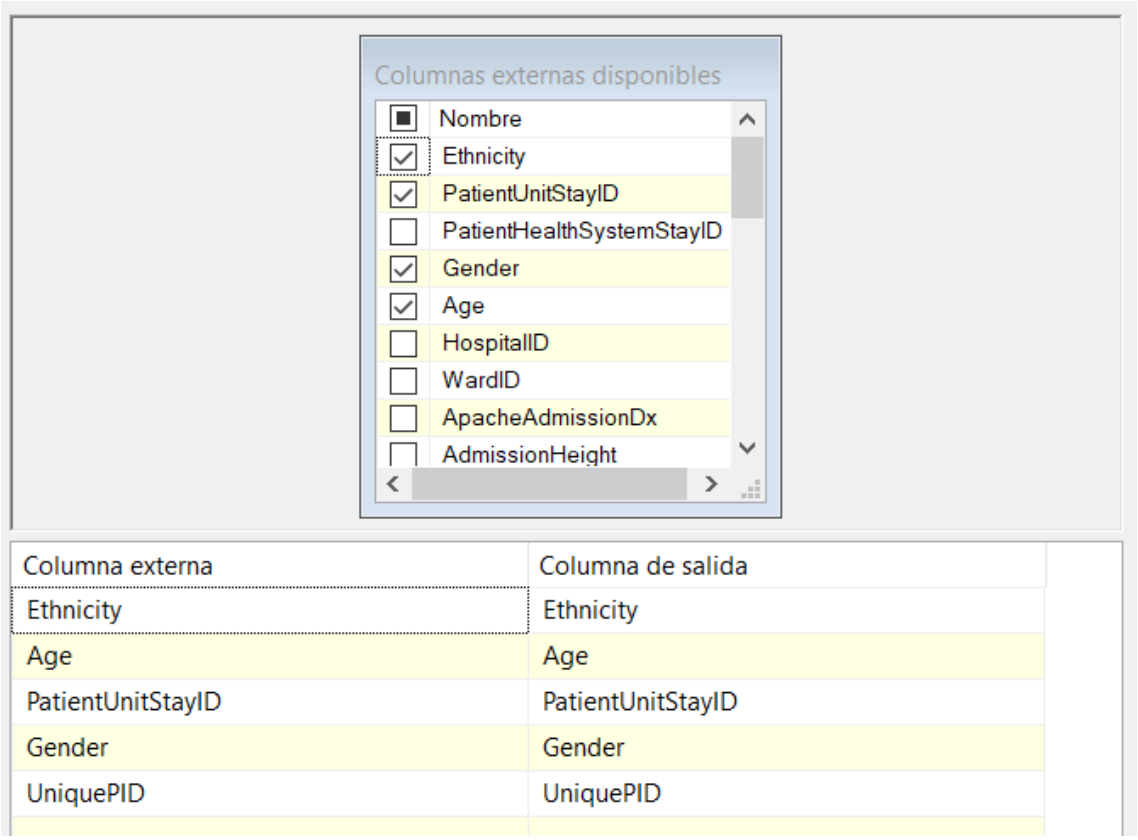

Figure 16: Carga pacientes BD

Posteriormente, se realizará un *lookup* para efectuar un *join* entre los campos `Ethnicity` y `Gender` de la base de datos **eICU**, y los campos `ethnicityString` y `genderString` obtenidos a partir de la siguiente consulta sobre nuestro **DW** (ver Figura 17):

```
SELECT * FROM Ethnicity, Gender
```

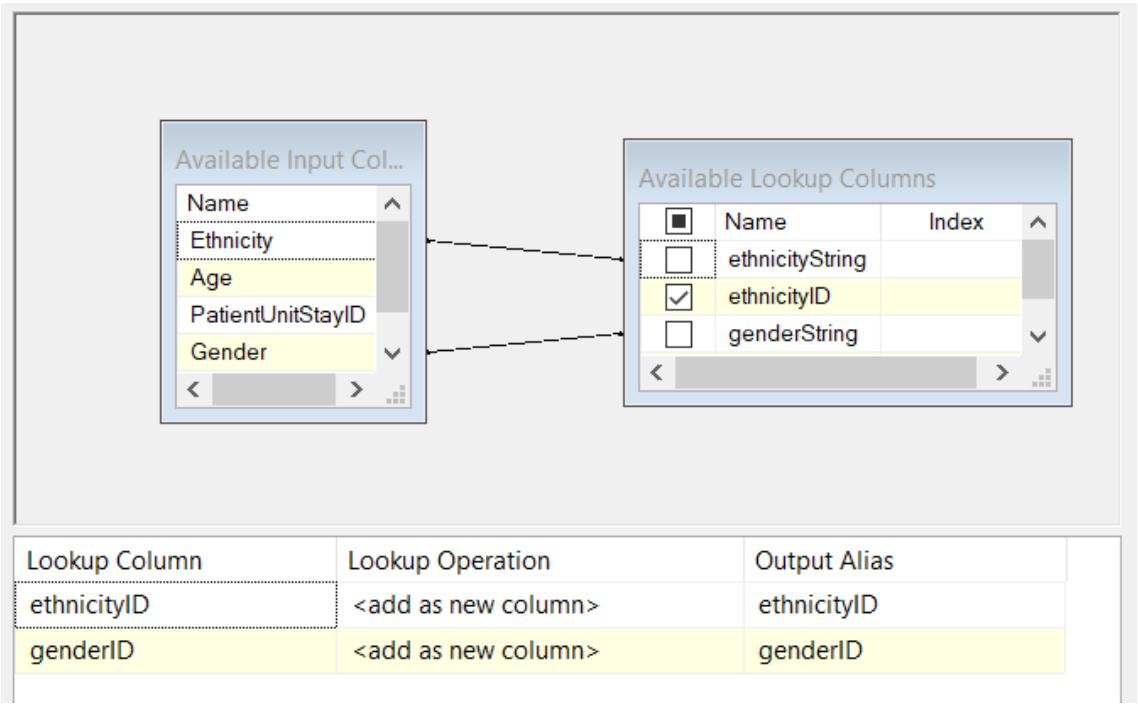

Figure 17: Busqueda_por_etnia_y_genero

Por último, se insertarán los siguientes atributos en el destino (ver Figura 18):

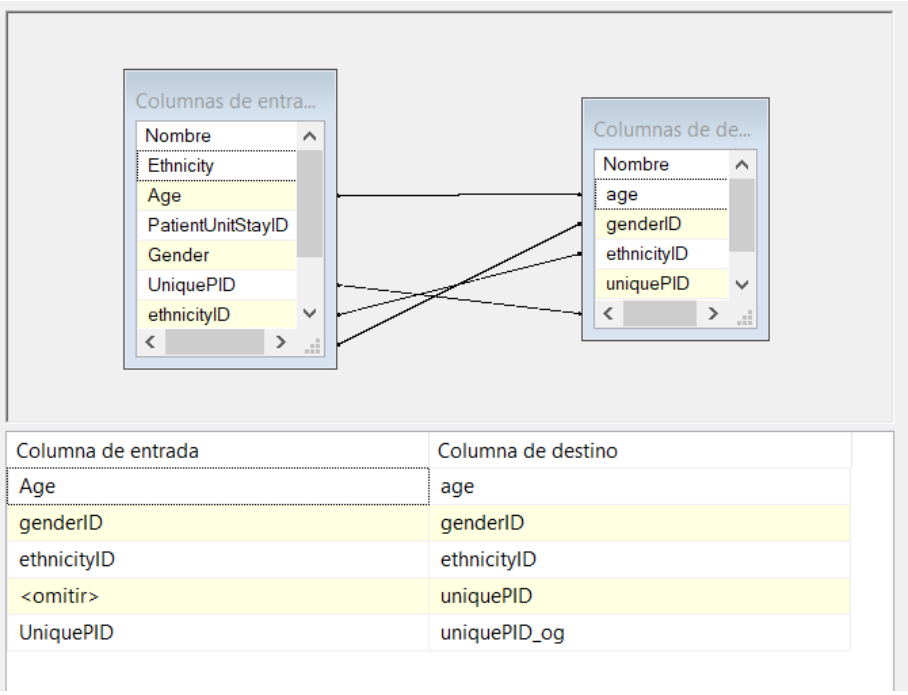

Figure 18: Pacientes_a_DW

Además, se incluirá el atributo `UniquePID`, ya que es necesario para permitir que `IngresoUCI` herede la clave foránea (`FK`) de `Paciente`.

6.10 Carga IngresoUCI

En este caso se necesita tener cargada la tabla Hospital, Paciente y tiempo (ver secciones 6.4, 6.8 y 6.9), y su flujo es el de la figura 19

Figure 19: Carga IngresoUCI

Para la carga de `IngresoUCI` aunque parezca muy simple, es debido al uso de consulta que interconectará todas las tablas como podrías, se utilizó como origen la siguiente consulta, mostrada en la Figura 20:

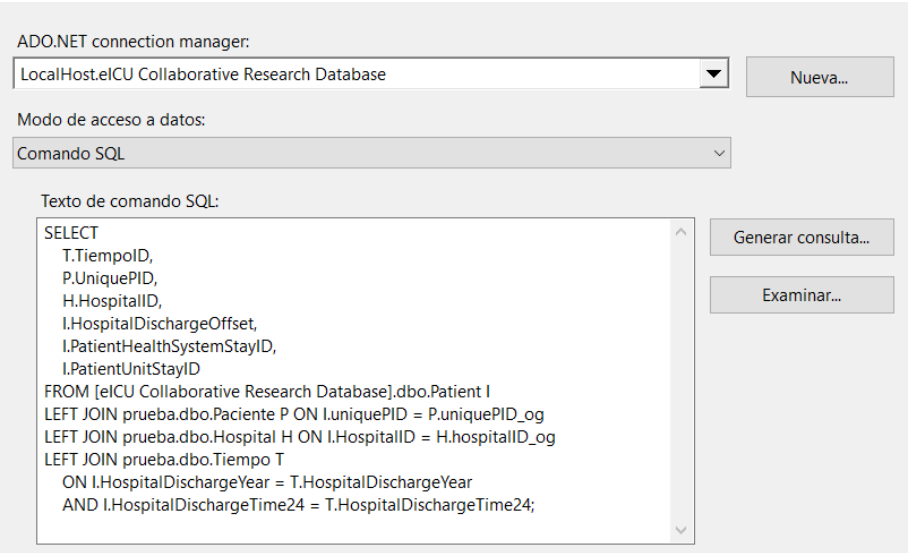

Figure 20: Paciente_en_BD

Nota: Cuando se menciona `prueba.dbo`, nos referimos a nuestra nueva base de datos/DW, ya que así la hemos denominado.

La consulta selecciona varios atributos de la base de datos **eICU Collaborative Research Database** y los combina con datos de nuestro **Data Warehouse (DW)**, específicamente de las tablas `Paciente`, `Hospital`, y `Tiempo`, utilizando *joins* (uniones).

Descripción de la consulta:

- **Campos seleccionados:**
  - `TiempoID`: Proviene de la tabla `Tiempo`, representando el identificador del tiempo del paciente.
  - `UniquePID`: Es el identificador único del paciente, que se obtiene de la tabla `Paciente`.
  - `HospitalID`: Representa el identificador del hospital, proveniente de la tabla `Hospital`.
  - `HospitalDischargeOffset`: Muestra el tiempo que ha pasado desde que el paciente fue dado de alta del hospital.
  - `PatientHealthSystemStayID`: Identificador de la estancia del paciente en el sistema de salud.
  - `PatientUnitStayID`: Identificador de la unidad en la que el paciente estuvo durante su estancia.
- **Proceso de unión (joins):** La consulta usa **LEFT JOIN** para combinar las tablas, lo que significa que se toman todos los registros de la tabla de la izquierda (de la base de datos `eICU`) y los registros coincidentes de las tablas de la derecha (de nuestro DW). Si no hay coincidencia, los valores de las tablas del DW serán `NULL`.
  - **Primer JOIN:** Se une la tabla `Patient` de `eICU` (aliased como `I`) con la tabla `Paciente` de nuestro DW (aliased como `P`), usando el campo `uniquePID` de ambas tablas. Aquí, se mapea el `uniquePID` de `eICU` al campo `uniquePID_og` de `Paciente` en el DW.
  - **Segundo JOIN:** Se une la tabla `Patient` de `eICU` con la tabla `Hospital` en nuestro DW (aliased como `H`), usando el campo `HospitalID`. Se compara `HospitalID` de `eICU` con `hospitalID_og` en `Hospital` en el DW.
  - **Tercer JOIN:** Se une la tabla `Patient` de `eICU` con la tabla `Tiempo` en el DW (aliased como `T`), utilizando los campos `HospitalDischargeYear` y `HospitalDischargeTime24` de `eICU` y comparándolos con los mismos campos en la tabla `Tiempo` de nuestro DW.
- **Resultado final:** La consulta extrae los datos de `eICU` y los enlaza con la información relevante de nuestro DW, lo que permite que los registros en la tabla `IngresoUCI` contengan las claves correctas de las tablas de nuestro DW (como `Paciente`, `Hospital`, y `Tiempo`).

Por último, se insertaron los datos en IngresoUCI de nuestro DW:

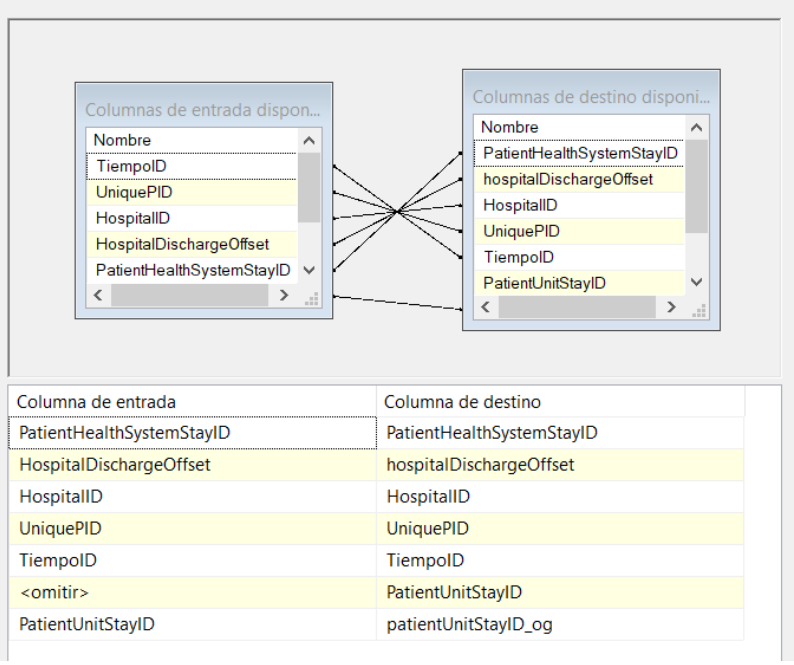

Figure 21: Ingreso_UCI_DW

Es importante destacar que la Carga IngresoUCI es de las tareas que más conexiones tiene debido a su importancia para mantener una lógica y conectividad clara en el trabajo.

6.11 Carga ApacheApsVar

Para el caso de la carga ApacheApsVar es necesaria que la carga IngresoUCI haya finalizado, esto debido a que será necesaria para que los datos obtenidos de la BD esten relacionados con el ingreso actual, para esto se usa el identificador de la base de datos original de diversas maneras para conectar la ApacheApsVar correspondiente al ingreso correcto. (ver figura 22)

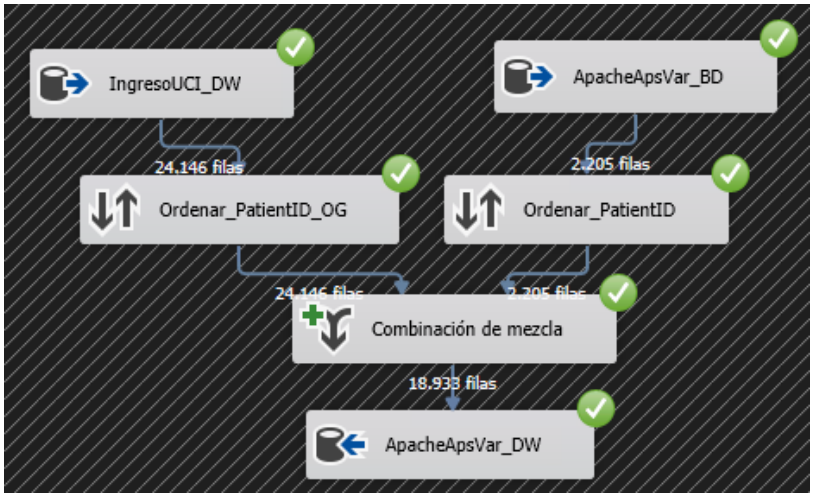

Figure 22: Carga ApsVar

Es importante destacar que este es un trabajo de índole **educativo**, por lo que se propusieron diversas maneras de obtener la **información de interés** a través del uso de varias **herramientas** en la carga. En esta ocasión, se realizó de manera similar a la carga de `ingresoUCI`, utilizando un `JOIN` para relacionar la información de dos tablas. Sin embargo, en lugar de usar consultas `SQL`, se conectaron la tabla `IngresoUCI` del **DW** y la tabla `ApacheApsVar` de la base de datos. Ambas se ordenaron según el identificador original de la base de datos (`patentUnitStayID_OG`), y los ingresos que fueron conectados con sus respectivos `apachesApsVar` se guardaron en el **DW** con el identificador actual (el autogenerado en la tabla `IngresoUCI`).

Cabe destacar que el **resultado** es correcto y fue comprobado: las **18,993 filas** corresponden a todos los ingresos que se vincularon con un `ApacheApsVar`, mientras que las restantes, que no fueron almacenadas, son ingresos sin información asociada a este campo.

6.12 Carga AdmissionDX

Parecido al caso de ApacheApsVar, trabaja de manera similar es un flujo de datos que se compila una vez se completa el ingresoUCI puede observar a detalle en la figura 23

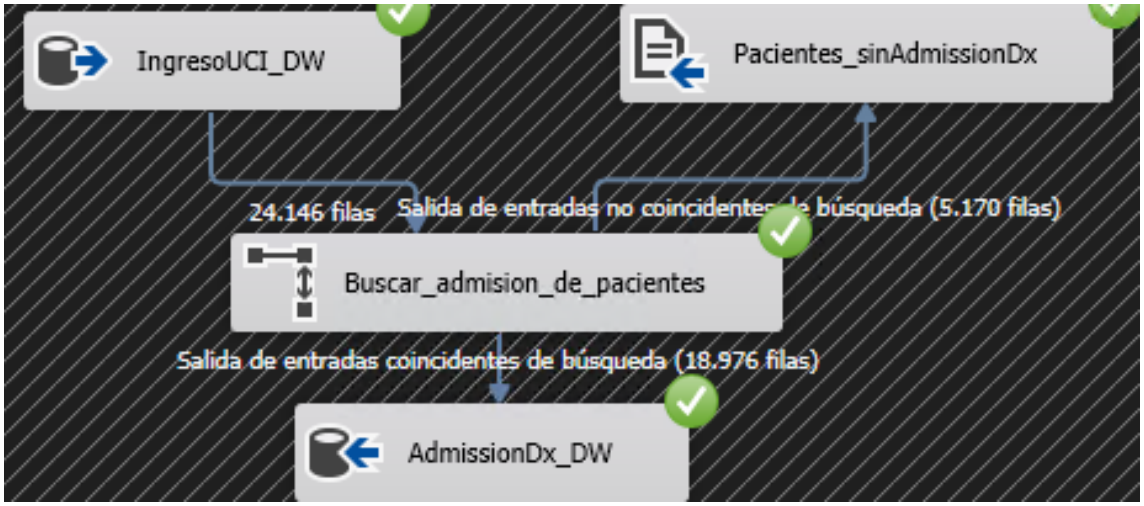

Figure 23: Carga AdmissionDx

Como se mencionó anteriormente, la intención de este trabajo es probar y llegar a soluciones a través de las diversas **herramientas** dadas. Aunque este caso podría ser exactamente igual al de `ApacheApsVar`, en este caso se utiliza únicamente la tabla `ingresoUCI`, de modo que mediante el bloque de búsqueda se guarden todas las filas de `admissionDX` de la base de datos **ElCU**. De manera similar a lo anterior, se usa la clave original (`patentUnitStayID_OG`) para vincular las admisiones con su respectivo ingreso. Finalmente, se guarda el identificador del ingreso (actual) de la base de datos, un identificador **autogenerado** para `AdmissionDX` y los atributos de interés de la base de datos (`AdmitDxEnteredOffset` y `AdmitDxName`).

Además, se puede observar el listado de **5,170 ingresos** que no tienen datos de admisión, los cuales fueron guardados en un archivo de texto plano para su posterior verificación.

6.13 Carga RespiratoryCare

Siguiento los inicios anteriores para empezar esta carga es necesario que la carga ingresoUCI finalice. Puede observar en la figura 24 el flujo de datos.

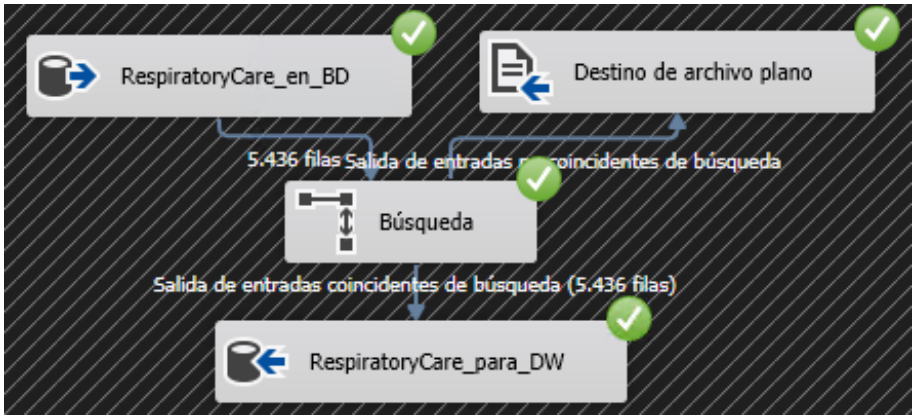

Figure 24: Carga RespiratoryCare

Este proceso es similar al de `ApacheApsVar`, pero cuenta con una ventaja visual al utilizar como origen la tabla `respiratoryCare` de la base de datos. Se obtienen **5,436 filas**, las cuales pasan por un bloque de búsqueda que se encarga de vincularlas con el ingreso mediante la clave `patentUnitStayID_OG`. Los datos resultantes se almacenan en el Data Warehouse, incluyendo el `PatientUnitStayID` (ID actual de ingreso), `VenStartOffSet`, `VentEndOffset`, `LowExhMVlimit` y `HiExhMVLimit`. Como es habitual, se generó de manera automática la clave `RespCareID`.

Es importante destacar que las cargas de `ApacheApsVar` (6.11), `Admission` (6.12) y `RespiratoryCare` (6.13) son procesos muy similares. Si adaptáramos cualquiera de estos tres flujos de datos para cada una de las respectivas cargas, se guardarían los mismos datos en el **DW**, lo que demuestra la gran versatilidad y flexibilidad que ofrece el proceso de **ETL**.

6.14 Carga Diagnosis

En el caso de diagnosis es la continuacion de carga Priority vista en la sección 6.5, el flujo de datos es la figura 25.

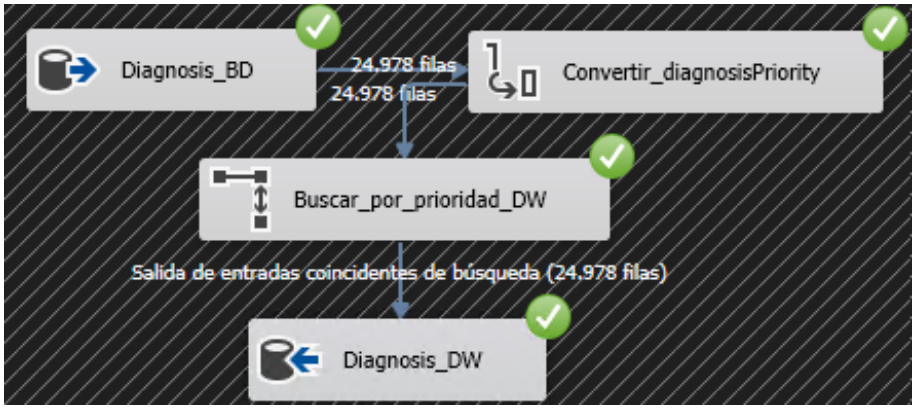

Figure 25: Carga Diagnosis

Para la carga de `Diagnosis`, se tomó como origen la tabla `Diagnosis` del **eICU**. Posteriormente, se realizó una conversión del atributo `DiagnosisPriority` a tipo cadena `[DT_STR]` con longitud 10.

A continuación, se realizará un *join* entre `Copy of DiagnosisPriority` y `priorityString` de la tabla `Priority` de nuestra base de datos/DW, y se seleccionará el `priorityID` para que pueda ser heredado por `Diagnosis` en nuestro **DW** (ver Figura 26).

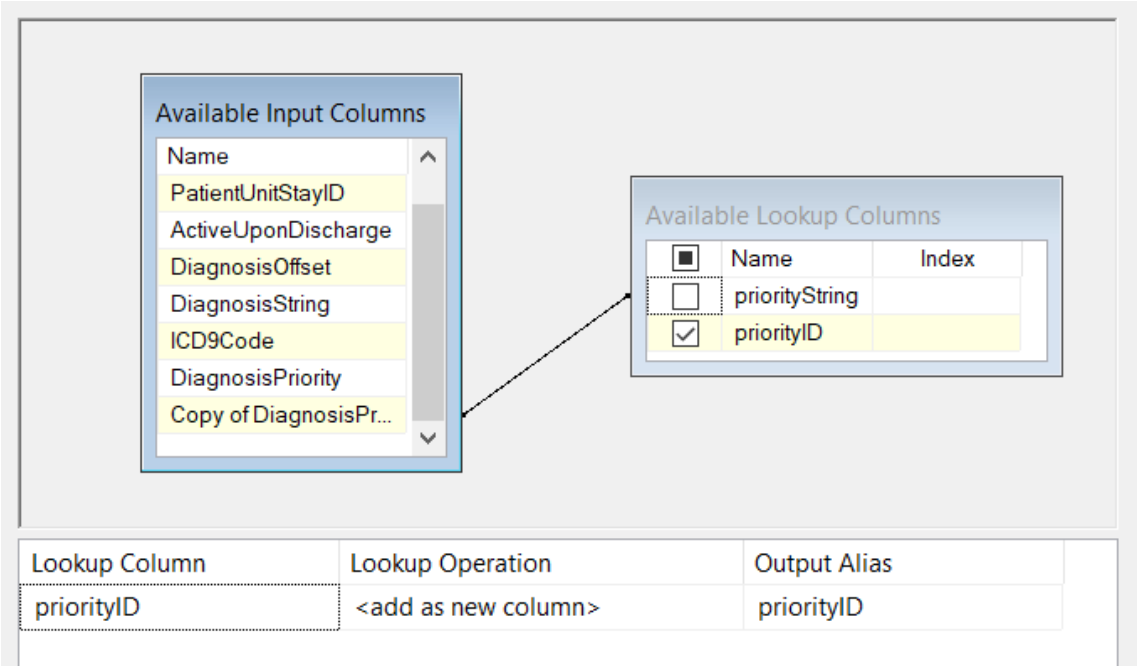

Figure 26: Buscar_por_prioridad_DW

Estos serán los atributos de `Diagnosis` en nuestro **DW**:

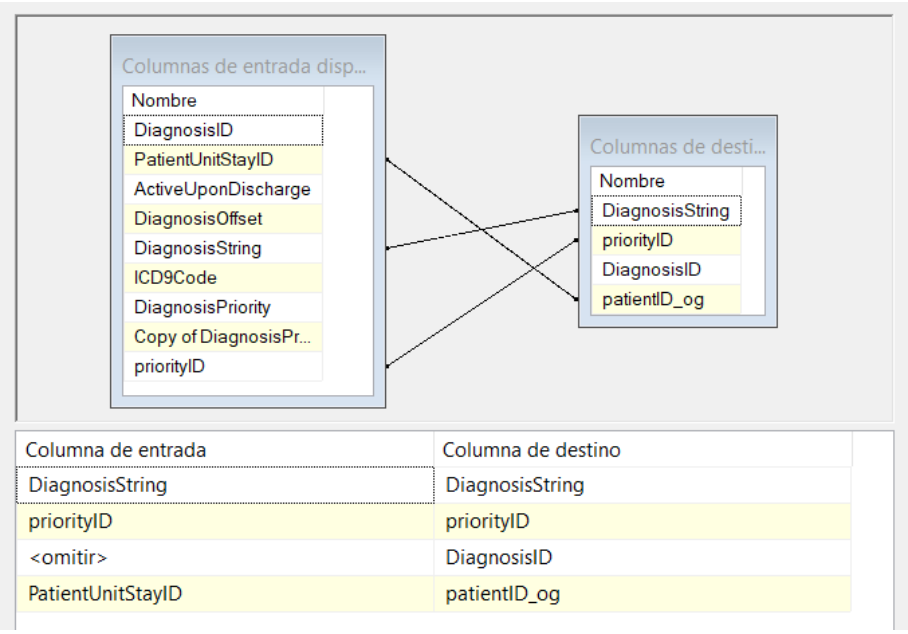

Figure 27: Diagnosis_DW

6.15 Carga UCI Diagnosis

En el caso de `UCI Diagnosis`, esta tabla corresponde a la relación **M:N** de dos tablas. Las tablas intermedias fueron nombradas con un prefijo `UCI` al principio, para destacarlas como tablas diferentes. En esta tabla se almacenan únicamente el `PatientUnitStayID` y el `DiagnosisID`, ambas claves **autogeneradas** y **originales del DW**. El flujo correspondiente a este proceso se muestra en la figura 28.

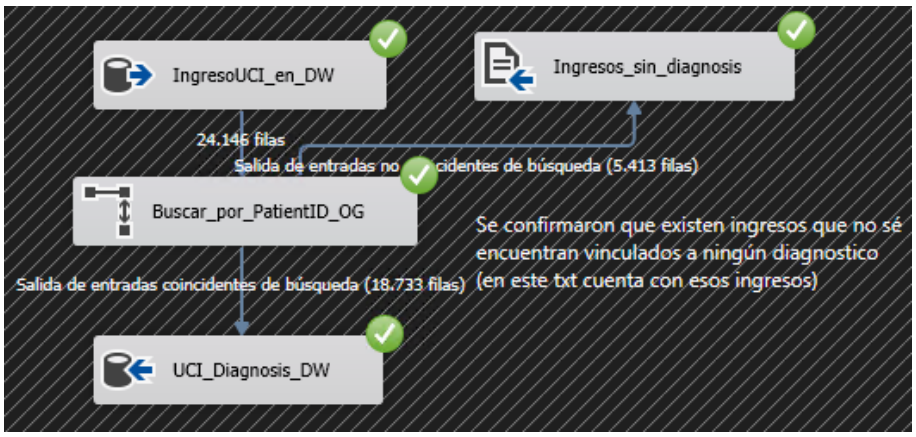

Figure 28: Carga UCI Diagnosis

Esta carga es relativamente simple, con la diferencia de que no se utilizan datos externos. En su lugar, se obtienen los ingresos del **DW** y, mediante un bloque de búsqueda, se filtra por el `PatientUnitStayID_OG`, relacionando los ingresos que tienen un `Diagnosis`. Estos se vinculan en el **DW**, donde solo se almacenan las claves de la tabla `IngresoUCI` (`PatientUnitStayID`) y la tabla `Diagnosis` (`DiagnosisID`).

6.16 Carga Alergia

Esta carga es la continuación de la carga tipo (sección 6.6), una vez compilado esa carga se procede a este flujo de datos (mirar figura 29)

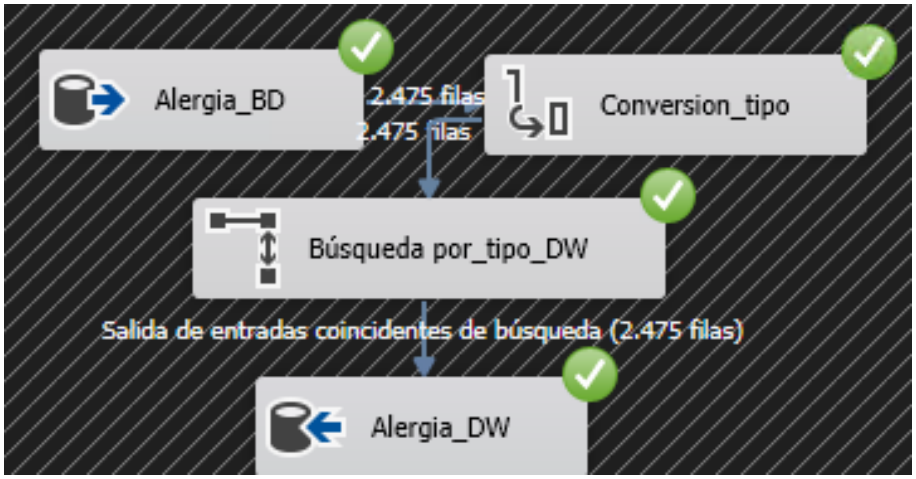

Figure 29: Carga Alergia

Al igual que en los casos anteriores, se obtiene la tabla `Alergia` de la base de datos **ElCU**. El atributo `AllergyType` se somete a un proceso de conversión debido a que no corresponde correctamente al formato del **DW**. A continuación, se utiliza un bloque de búsqueda para vincularla con la tabla `Tipos`, para finalmente guardar los datos en el **DW**.

6.17 Carga UCI Alergia

Siendo nuevamente una tabla de relación M:N, necesita la carga de Alergia e IngresoUCI para empezar. Su flujo es el siguiente (ver figura 30).

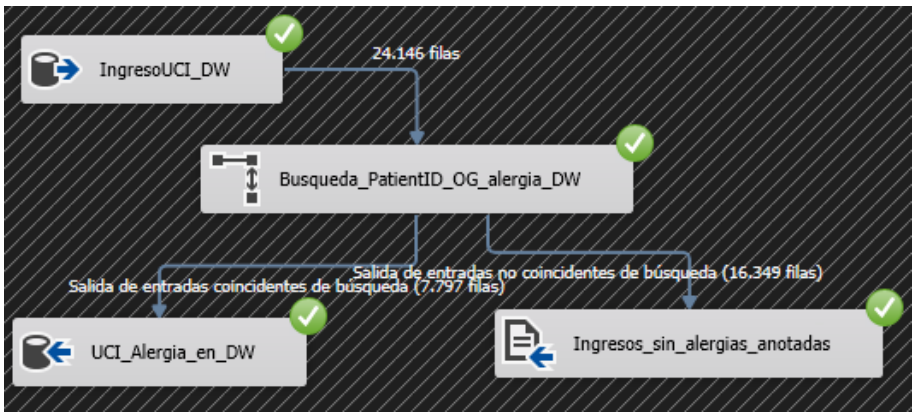

Figure 30: Carga UCI Alergia

Donde se obtienen los ingresos del DW y se conectan mediante un bloque de Búsqueda a la tabla alergia, usando el `PatientUnitStayID_OG` para hacer la conexión.

6.18 Carga Medicacion

En el caso de medicacion es la continuacion de carga RouteAdmin (Seccion 6.7) , el flujo de datos es la figura 31.

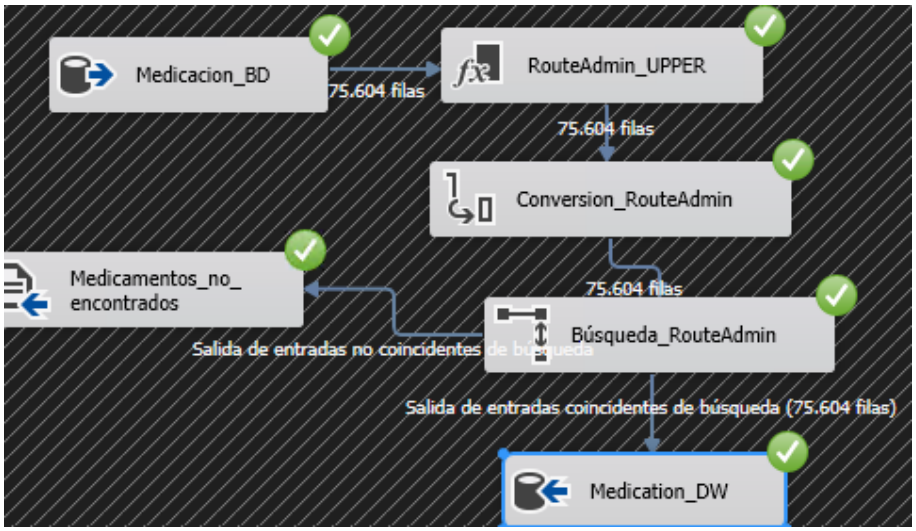

Figure 31: Carga Medicacion

Al igual que en la carga `RouteAdmin`, se utilizó un bloque de columna derivada, ya que es igualmente necesario por la misma razón. Al haber convertido todos los valores de `RouteAdmin` a mayúsculas, actualmente, si intentamos conectar `RouteAdmin` con `Medicación`, algunas filas no se vincularán debido a que la ruta "Oral" en el almacén está registrada como "ORAL". De esta manera, se realiza una conversión de datos previa a la búsqueda para evitar errores de tipo de datos en el `RouteAdmin` recién modificado. Una vez que se conecten los datos de `Medicación` con la tabla `RouteAdmin`, estos se almacenarán en el **almacén**.

6.19 Carga UCI Medicacion

Para finalizar, se carga la última tabla, `UCI Medicación`, la cual se creó para representar la conexión M:N entre `Medicacion` e `IngresoUCI`. El flujo de datos correspondiente se ilustra en la figura 32.

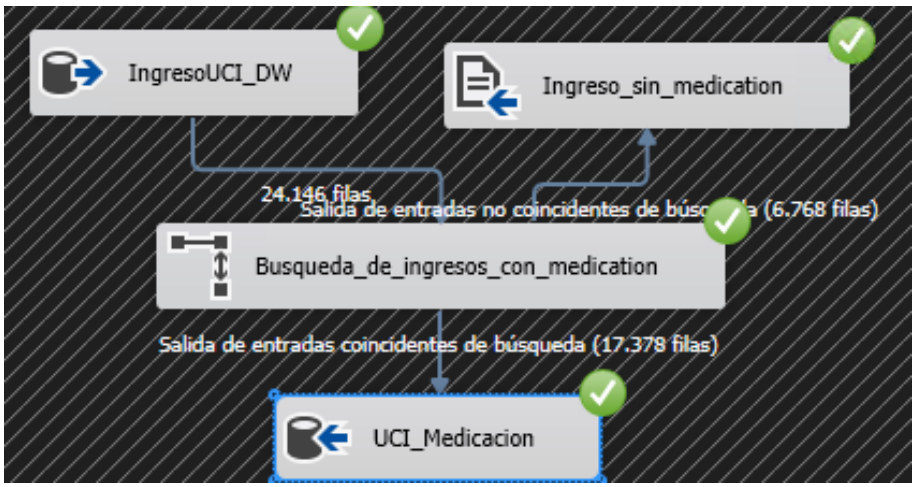

Figure 32: Carga UCI Medicación

Al igual que en los otros casos de tablas intermedias, en este proceso se buscan los ingresos desde el almacén y, mediante un bloque de búsqueda, se obtienen las claves conectadas de `Medicacion` e `IngresoUCI`.

7 Dificultades encontradas

Una de las principales dificultades fue la complejidad de la base de datos eICU, que incluye una gran cantidad de tablas y atributos. Esto exigió un análisis detallado para identificar las tablas y campos clave en un modelo centrado en pacientes con enfermedades respiratorias. Además, enfrentamos problemas de permisos al intentar visualizar el modelo relacional en SQL Server, lo que requirió modificar las autorizaciones del propietario de la base de datos para acceder a los diagramas de relación.

Además el comienzo del trabajo podría ser lo más angustioso, al tener tanta información y opciones llega a ser un poco abrumador, desde la selección de una población concreta y modelar un almacén para dicha población termina dejando muchas dudas sobre cuantas tablas es esperable eliminar, si se esta simplificando de más o se esta tomando una decisión que afectará los siguientes apartados.

También se tuvo ciertos problemas con la máquina virtual usada al quedarse pillada a la hora de iniciar, se tuvo que reiniciar lo que conllevo a la perdida de datos no guardados en otros medios.

8 Conclusión

El proceso ETL (Extract, Transform, Load) es fundamental en la construcción y mantenimiento de un almacén de datos eficiente. Permite la integración de datos provenientes de diversas fuentes, transformándolos en información útil y cargándolos en una estructura de almacenamiento adecuada. Este proceso no solo garantiza la calidad y coherencia de los datos, sino que también facilita su análisis y explotación posterior, convirtiéndose en una pieza clave para la toma de decisiones en entornos empresariales y científicos.

Visual Studio es una herramienta altamente versátil que facilita la creación de procesos ETL complejos mediante bloques de herramientas visuales. Su interfaz intuitiva permite integrar diversas técnicas de procesamiento de datos de manera modular, lo que facilita la búsqueda de soluciones alternativas para abordar cada carga de datos de manera eficiente. Mediante su uso, es posible conectar diferentes fuentes de datos, aplicar transformaciones necesarias y cargar la información en el destino deseado sin la necesidad de escribir extensas líneas de código, lo que reduce el tiempo de desarrollo y mejora la accesibilidad del proceso.

Finalmente, este trabajo ha demostrado que, aunque los procesos ETL pueden ser complejos y desafiantes, herramientas como Visual Studio, junto con un enfoque estructurado, pueden simplificar la tarea de integrar datos de diversas fuentes. Además, la implementación adecuada de estas soluciones no solo mejora la calidad de los datos almacenados, sino que también permite una mayor flexibilidad y adaptabilidad frente a futuras modificaciones o expansiones del sistema.

9 Github y conjunto de instrucciones para su correcto despliegue en SQL Server.

Todo el proyecto está accesible en github [2] donde se detalla más específicamente como desplegar en SQL.

References

[1] eICU Collaborative Research Database. *eICU Collaborative Research Database*. `https://eicu-crd.mit.edu/about/eicu/`. Accessed: 2024-11-14. 2024. URL: `https://eicu-crd.mit.edu/about/eicu/`.

[2] Diegodepab. *Almacén UCI Sanitaria*. `https://github.com/Diegodepab/almacen_UCI_Sanitaria`. Accessed: 2024-11-14. 2024. URL: `https://github.com/Diegodepab/almacen_UCI_Sanitaria`.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA