Daniela Castaño Medina - A00400872

Davinson Alexander Arteaga Bermúdez - A00419669

Gian Alepsi Mendoza Oviedo - A00419307

Juan José Villegas Triana - A00422503

1. Introducción

El objetivo de este proyecto fue construir un proceso ETL en **Azure Data Factory (ADF)** para integrar múltiples fuentes de datos heterogéneas relacionadas con el conjunto de propiedades de **Ames Iowa**, y generar como resultado un archivo S**`alida.csv`** listo para análisis y modelamiento predictivo.

Las fuentes de información incluían:

- Una **fuente relacional** en PostgreSQL (base de datos `amesDB` en Neon) con información estructurada sobre características constructivas y de venta de las propiedades.
- Varias fuentes **no relacionales** en **MongoDB Atlas** (`garage`, `pool`, `bsmt`, `misc`) con atributos complementarios almacenados como documentos.
- El archivo plano **`AmesProperty.csv`** con información general de cada propiedad.

Estas fuentes presentaban diferencias en formato (CSV, tablas relacionales, documentos JSON), esquemas, tipos de datos y manejo de valores faltantes, por lo que era necesario diseñar un proceso ETL que:

1. Ingiriera la información desde las distintas tecnologías (PostgreSQL, MongoDB, Blob Storage).
2. Estandarizara nombres de columnas, tipos de datos y reglas para valores nulos.
3. Integrará todos los atributos a nivel de propiedad mediante la llave **`PID`**.
4. Aplicara las transformaciones requeridas por el enunciado (cálculo de variables, tratamiento de fechas, ajustes de `NA` y `0`, etc.).
5. Generará un archivo único **`Salida.csv`** almacenado en un **Data Lake** para su posterior uso en tareas de analítica y ciencia de datos.

Para lograrlo se utilizó **Azure Data Factory** como orquestador y motor de transformación, en conjunto con **Azure Data Lake Gen2**, **PostgreSQL en Neon** y **MongoDB Atlas**, incorporando además el control de versiones del proyecto mediante la integración con **GitHub Classroom**.

2. Actividades realizadas

La configuración inicial del proyecto consistió en crear la fábrica de datos ***Proyecto-TI-2025-2*** en la región *mexicocentral*, habilitando la identidad administrada e integrando la solución con el repositorio de **GitHub** del curso. A partir de esta base, se definieron los linked services necesarios para conectar las diferentes tecnologías utilizadas en el proceso. Entre ellos, se configuraron ***NeonConn*** y ***ClientesPostgreSQL***, ambos orientados a establecer la comunicación con las bases ***PostgreSQL*** alojadas en Neon, que funcionaron como fuente relacional principal. También se creó el linked service *MongoConn*, encargado de conectar ADF con la base *test* de ***MongoDB Atlas***, que contenía las colecciones ***garage***, ***pool***, ***bsmt*** y ***misc***. Finalmente, se definió *DataLakeFinal*, un linked service parametrizado sobre Azure Data Lake (BlobFS), utilizado para almacenar el archivo final ***Salida.csv*** y que incorpora parámetros para flexibilizar la ruta y nombre del archivo resultante.

Con estos servicios creados, se configuraron los datasets necesarios para representar cada fuente y destino. El dataset ***amesproperty*** referenció el archivo *Ames Property.csv*, que contiene la información general inicial de cada propiedad. El dataset ***AzurePostgreSqlTable1*** apuntó a la tabla/vista disponible en *amesDB*, donde se consolidó la información relacional del modelo Ames. Para las fuentes no relacionales, se construyeron los datasets ***bsmtentrada***, ***garageentrada***, ***poolentrada*** y ***miscentrada***, orientados a extraer la información directamente desde MongoDB. Paralelamente, se definieron los datasets *bsmtmongo*, *garagemongo*, *poolmongo* y *miscmongo*, que almacenan en formato delimitado los extractos intermedios de estas colecciones dentro del Data Lake. Por último, se creó el dataset ***Salida***, encargado de definir la estructura del archivo final *Salida.csv*. Esta organización permitió separar claramente las fuentes originales, los extractos intermedios y el artefacto de salida del ETL.

Para transformar las colecciones de MongoDB en un formato adecuado, se construyó un pipeline específico llamado ***mongocsv***. Este pipeline empleó varias **actividades *Copy*** con una estructura común: cada actividad utilizó como origen un *MongoDbAtlasSource* conectado mediante el linked service *MongoConn* y como destino un ***DelimitedTextSink*** basado en *AzureBlobFSWriteSettings*. Este sink se configuró con ***FlattenHierarchy*** para aplanar correctamente la estructura JSON y generar archivos *.CSV*. Como resultado, se obtuvo una versión tabular de cada colección de MongoDB, almacenada como archivos delimitados (bsmtmongo, garagemongo, poolmongo y miscmongo), que posteriormente serían consumidos por el data flow principal.

Antes de integrar las diversas fuentes, fue necesario aplicar transformaciones de limpieza, especialmente para manejar valores faltantes y estandarizar la estructura de datos. Se realizaron ajustes de nulos tanto de manera individual por fuente como posteriormente, mediante una derivación global aplicada al conjunto integrado. Esto permitió cumplir la regla del proyecto: reemplazar nulos por *0* en variables numéricas y por *"NA"* en variables categóricas. Esta etapa fue clave para garantizar coherencia entre las fuentes y evitar problemas durante los procesos de join e integración, especialmente en atributos provenientes

de colecciones de MongoDB, donde no todas las propiedades presentan información asociada (como piscinas o garajes).

El corazón del proyecto fue el ***Mapping Data Flow*** denominado ***dataflow1***, ejecutado desde el pipeline ***pipeline1***. En este flujo se integraron simultáneamente seis fuentes: el archivo de propiedades generales, la tabla del modelo relacional de PostgreSQL y los cuatro extractos de MongoDB ya convertidos a formato delimitado. A partir de estas fuentes, se inició una serie de transformaciones destinadas a normalizar y renombrar columnas mediante operadores *Select*. Este proceso permitió alinear los esquemas, eliminar inconsistencias y adaptar los nombres de columnas al formato solicitado en la salida. Asimismo, varias transformaciones complejas, como la suma de áreas para obtener *GrLivArea*, la consolidación de baños y habitaciones por piso, o la separación del mes y año de venta desde la fecha original, se resolvieron previamente en el origen relacional mediante una **consulta SQL**, simplificando el data flow y garantizando mayor claridad en la estructura final.

Posteriormente, se realizó la fase de integración mediante una secuencia de ***Left Join*** sobre la columna ***PID***. A través de estos joins se incorporó la información de las colecciones de MongoDB, procurando siempre mantener la totalidad de las propiedades presentes en la base relacional, aun cuando algunas carecieran de elementos como garaje, piscina o características misceláneas. Una vez integradas todas las fuentes no relacionales, se efectuó un ***Inner Join*** con los datos provenientes del archivo CSV original, con el objetivo de evitar duplicados y asegurar que el modelo final incluyera únicamente propiedades válidas y coincidentes en ambas fuentes principales.

Después de integrar toda la información, se aplicó un ***sort*** por *PID* para garantizar un orden coherente en las propiedades, lo que también facilitó validar manualmente el correcto funcionamiento del *Inner Join*. Finalmente, el resultado del flujo se dirigió al sink asociado al dataset *Salida*, donde se generó el archivo *Salida.csv* dentro del Data Lake. Este archivo consolidó para cada propiedad todas las columnas exigidas por el enunciado: datos generales, información relacional, atributos del sótano, garaje, piscina, características misceláneas y detalles de venta.

3. Lecciones Aprendidas

A lo largo del desarrollo del proyecto se identificaron múltiples lecciones relevantes tanto desde el punto de vista técnico como conceptual. Una de las primeras fue comprender la importancia del diseño adecuado de la capa de conexiones y datasets. Configurar correctamente los *linked services* para **Data Lake Gen2**, **PostgreSQL** y **MongoDB** resultó fundamental para evitar errores de autenticación, problemas de acceso y fallas en la lectura o escritura de datos. Asimismo, separar los datasets en categorías —fuentes originales, extractos intermedios desde Mongo y dataset final— permitió mantener una arquitectura clara

y modular, facilitando la comprensión del flujo ETL y permitiendo reutilizar artefactos sin generar duplicaciones innecesarias.

Otro aprendizaje clave estuvo relacionado con el manejo de datos semiestructurados provenientes de MongoDB. Las colecciones almacenaban su información en formato documento JSON y, en muchos casos, con valores numéricos representados como `"$numberLong"` o con estructuras anidadas, con esto se evidenció lo importante que es la estandarización de nombres y tipos de datos. La presencia de columnas con espacios, mayúsculas y nombres inconsistentes —como “Year Remod/Add”, “foundation” o “functional”— hacía necesaria una etapa específica de renombrado mediante transformaciones *Select*.

Asimismo, la existencia de diferentes representaciones de valores nulos entre las fuentes llevó a establecer una regla unificada: reemplazar nulos por *0* en columnas numéricas y por *“NA”* en columnas cualitativas. Esto se implementó de forma centralizada en la transformación *derivedColumn1*, lo que evitó discrepancias y aseguró coherencia en la salida final. El diseño de los *joins* dentro del Data Flow fue otra fuente de aprendizajes. Integrar varias fuentes mediante una cadena de *left joins* y rematar con un *inner join* obligó a trabajar con sumo cuidado en la administración de llaves como `PID`, `PID_ames` y los identificadores específicos de cada colección (*PID_GARAGE*, *PID_POOL*, etc.). Además, fue evidente que el orden de los joins y la limpieza posterior de duplicados mediante transformaciones como *select1* afecta directamente la claridad, consistencia y estructura del resultado. Una mala secuencia de joins podía llevar a duplicaciones, registros incompletos o incluso a errores de esquema al momento de ejecutar el sink.

Finalmente, el uso de **GitHub** mediante GitHub Classroom facilitó enormemente el trabajo colaborativo. La integración de **ADF** con control de versiones permitió mantener un historial claro de cambios, organizar el trabajo del grupo mediante ramas y commits, y versionar adecuadamente pipelines, dataflows, datasets y definiciones de linked services. Esto dejó en evidencia la importancia de combinar herramientas de integración continua con plataformas de orquestación de datos.

En este proceso también surgieron errores reales que se convirtieron en aprendizajes valiosos. En una ocasión, debido a una falla en la sincronización entre Data Factory y GitHub, se eliminó accidentalmente una rama perteneciente a uno de los compañeros, lo que ocasionó la pérdida temporal de su trabajo. Esta situación permitió comprender la importancia de manejar con cuidado las ramas, evitar sincronizaciones impulsivas y validar siempre el estado del repositorio antes de realizar un **Pull Request** en la plataforma.