****

**DISEÑO E IMPLEMENTACIÓN DEL MODELO RELACIONAL DE LA BASE DE DATOS PARA EL SERVICIO AUTÓNOMO DE LA PROPIEDAD INTELECTUAL (SAPI)**

**Profesor:**

Fernando Torre Mora

**Equipo:**

David Díaz. 20-10019

Ian García. 19-10087

**Introducción**

En un documento anterior, titulado “…”, se discutió y presentó el modelo teórico de la base de datos que se diseñó para el registro de solicitudes de marcas del Servicio Autónomo de la Propiedad Intelectual (SAPI). En un primer vistazo, se tomaron múltiples decisiones referentes a cómo debía estar constituida la base de datos y se presentó un diagrama ER adjunto a descripciones del mismo, identificando aspectos claves de funcionamiento. Sin embargo, inherente al propio progreso de aprendizaje del sistema del SAPI, hemos tenido que modificar algunos aspectos del funcionamiento del modelo presentado anteriormente.

Aprovechamos la entrada a una fase de planteamiento e implementación para terminar de corregir aquellos aspectos que no satisfacían las solicitudes del ente (SAPI) o que, por alguna razón, podían complicar la implementación de la base de datos.

Este proyecto pretende ser de utilidad exclusivamente para el Servicio Autónomo de la Propiedad Intelectual (SAPI); por lo que ni el diseño conceptual ni relacional de la base de datos es extensible a otros proyectos, incluso si los mismos están relacionados con de alguna manera con el SAPI, pues este modelo se basa en partes muy específicas del funcionamiento del SAPI e incluso el orden de las planillas del mismo.

La tecnología principal utilizada en la implementación del proyecto fue PostgreSQL; sin embargo, en menor manera se utilizó Python para la creación de ciertos scripts que serán mencionados en secciones posteriores del informe. Con respecto a la creación del informe en sí, que forma parte de lo implementado, se utilizó únicamente Microsoft Word.

**Modelo físico de la base de datos**

Para poder llevar a cabo la implementación de la base de datos, primero era necesario obtener el modelo relacional de la misma a partir de un modelo conceptual, ya creado con anterioridad y aprobado por el SAPI.

Para ello, se hicieron traducciones del esquema E-R-E previamente presentado a un esquema UML. En principio, simplemente se tradujo de la forma más natural y simple posible, sin tomar en cuenta demasiados detalles.

En una etapa posterior se añadieron atributos extra, principalmente aquellos que son claves postizas, ya que UML exige la creación de una clave primaria por cada una de las clases sin excepción.

En la etapa de recolección y análisis de datos, nos dimos cuenta de marcadas diferencias en la manera en la que se reciben los datos con respecto a la manera que intuíamos que se recibirían (véase la sección XX); por lo cual, en esta etapa sí hubo cambios significativos en el diagrama UML, que serán explicados de una forma más detallada en la sección XX después de la presentación del mismo diagrama.

Este hecho no solamente supone un cambio en el diagrama UML, sino que (bajo nuestra consideración) supone una disminución significativa en la calidad de la base de datos. Recomendamos mejorar el formato del archivo de entrada y analizar el orden en que son presentados los datos, puesto que no están ordenados de manera que facilite la rápida población de la base de datos-

**Diagrama UML asociado a la base de datos**

APODERA

Número Poder:

PROPIEDAD COLECTIVA

Tipo:

PRIVADO

Origen:

PÚBLICO

Pertenencia del estado:

MIXTO

Nombre:

DENOMINATIVO

Nombre:

GRÁFICO

Imagen:

SIGNO

ID\_signo:

Edad:

Nombre:

Dirección:

Marca:

SIMBOLIZA

MARCA

ID\_marca:

Edad:

Nombre:

Dirección

POSEE

NATURAL

Edad:

Nombre:

Dirección:

Solicitante:

JURÍDICO

Edad:

Razón social:

Dirección

Solicitante:

REGISTRA

PAÍS

Nombre:

SOLICITANTE

ID\_solicitante:

Documento:

Tipo:

Número:

Nombre:

Correo:

Teléfono:

Fax:

REPRESENTANTE

Documento:

Nombre:

APODERADO

Numero agente:

Edad:

Nombre:

Dirección

SOLICITA

Hoja:

Solicitud:

Solicitante:

PRIORIDAD EXTRANJERA

Número:

Fecha:

PRIORIZA

Hoja:

Prioridad:

Solicitud:

SOLICITUD

Numero\_solicitud:

Fecha:

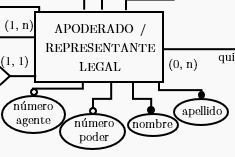
Numero\_tramite:

Numero\_referencia:

Marca:

**Reporte de cambios del diagrama UML con respecto**

* En el archivo de entrada se identificaba a los solicitantes por una clave postiza llamada *id\_solicitante*, cosa que nunca se supuso, debido a que se esperaba que fueran identificados por su documento de identidad. Como resulta lógico, no nos imaginamos que los solicitantes podrían llegar a presentar una solicitud sin incluir su documento de identidad en los datos.
* Anteriormente, los apoderados y representantes legales se encontraban en una misma clase; sin embargo, debido a la distinción clara que se hizo de ellos en el documento de entrada, se decidió optar por apartarlos en clases diferentes, una clase APODERADO para los apoderados y una REPRESENTANTE para los representantes legales.
* Anteriormente, un apoderado se relacionaba con un solicitante a través de la solicitud en la cual dicho apoderado representaba al solicitante; sin embargo, debido a la propia naturaleza del documento de entrada de datos se forzó a tener una conexión directa entre un apoderado y un solicitante.
* Por la manera en la que se recibieron los datos, también se forzó a que un solicitante tuviera solo un apoderado, cosa que no representa la situación real y que entonces tuvimos que solucionar. La solución supuso la creación de una clase intermedia llamada APODERA, lo mismo ocurría con REPRESENTANTE, cuya solución fue análoga.
* En el modelo conceptual se había considerado el número de poder de un apoderado como parte de sus atributos, véase la Figura adjunta

****

Esto, claramente, forzaba el hecho de que un apoderado no puede tener más de un número de poder, lo que claramente no es realista, pues un apoderado tiene tantos números de poder como personas jurídicas esté representando. Por lo tanto, se decidió incluir el número de poder a la relación entre APODERADO y SOLICITANTE.

**Datos de la base de datos**

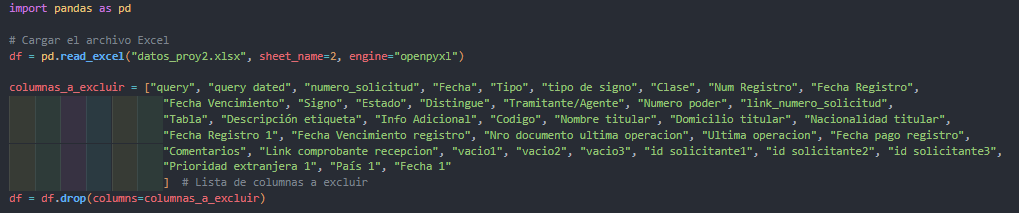
Esta fue probablemente la sección más conflictiva y tediosa de todas, se comenzó simplemente ojeando cómo eran presentados los datos y empezamos por tantear distintos planes para la inserción. Sin embargo, rápidamente nos conseguimos con problemas que serían algo difíciles de solucionar, a continuación, enunciaremos algunos de ellos, indicando su respectiva solución

* Existía una solicitud con número de solicitud repetido, simplemente compartían todos los datos, por lo que se notaba que fue un error de doble inscripción de la información que, en principio, se resolvía tan solo eliminando uno de los registros. Sin embargo, rápidamente encontramos que una de las solicitudes era procesada por un solicitante, mientras que la otra (repetida) era procesada por un solicitante distinto. Esto nos llevó a tener que eliminar la solicitud repetida, pero agregar al solicitante de la segunda como segundo solicitante de la primera.

Más allá de esto, se consiguió crear los scripts de INSERT a través de una librería de Python llamada pandas. Esta misma, facilita el proceso de lectura de la información de un archivo .xlsx (formato en que fueron entregados los datos para la población de la base de datos).

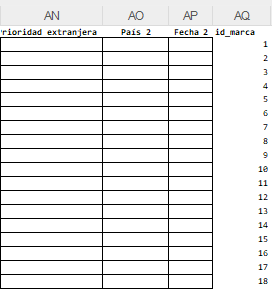
Aunado a esta, usamos también algunas herramientas integradas en Visual Studio, como el reemplazo de todas las iteraciones de una palabra, que permitían cambiar nombres fácilmente, e incluso reformatear el archivo antes generado por pandas en Python.

Sin embargo, el proceso pasó a ser bastante tedioso al darnos cuenta que pandas reconocía automáticamente algunas columnas que estaban ocultas en el archivo .xlsx. Por lo que explícitamente se le indicaba a pandas qué columnas debía evitar leer, teniendo muchas veces que escribir a mano una a una, como se muestra en la figura adjunta.



Para cada uno de los distintos *insert* se requería de un script de Python distinto que, usando pandas, fuera capaz de crear la instrucción. Resultaba por ser un proceso bastante manual, donde se debía indicar las columnas a incluir, el debido formateo de los datos y alguna consideración más.

Decidimos agregar también una columna al archivo de entrada llamado id\_marca, que sería usado para asegurar las conexiones entre las solicitudes y las marcas

****

Se tuvo que generar un archivo insert varias veces en múltiples ocasiones, por ejemplo, se generó un insert para cada uno de los solicitantes 1 asociados a una solicitud, se adjuntó la data de otro insert que contenía los solicitantes 2 y posteriormente lo mismo con solicitantes 3, en el resultado final usted verá un único insert, pero este se generó a partir de unir los tres insert de los tres solicitantes. Esto, por supuesto, no fue algo exclusivo de los solicitantes, sino que también ocurrió con las prioridades extranjeras, los países y algunos otros atributos.

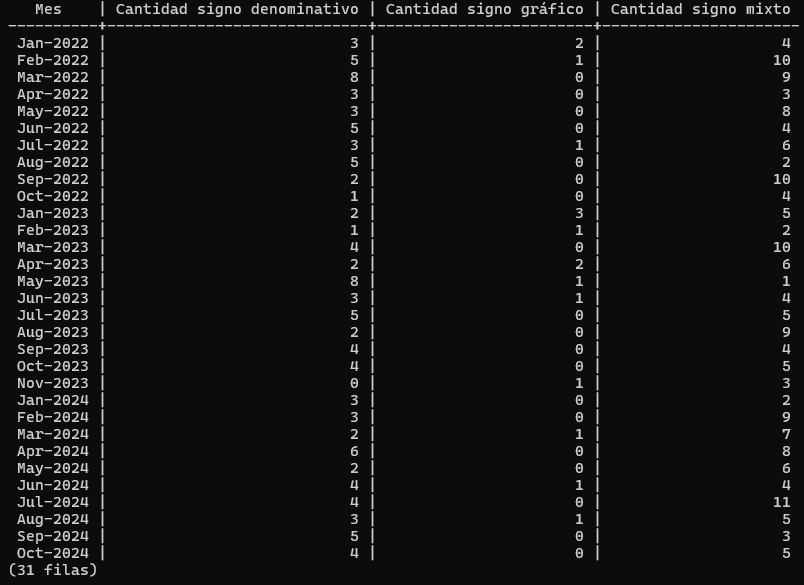
Otro problema relacionado con el proceso de inserción de datos ocurrió al darnos cuenta que no se describe el tipo para las personas jurídicas, lo que no solamente plantea una situación irrealista, sino que condiciona nuestra especificación del modelo conceptual, puesto que habíamos considerado el tipo como un atributo no nulo (tal como se especificaba en la planilla de ejemplo que nos facilitaron), pero al no presentarse en esta tabla de datos, entonces tuvimos que cambiarlo a un atributo que admite NULL.

**Pruebas y resultados**

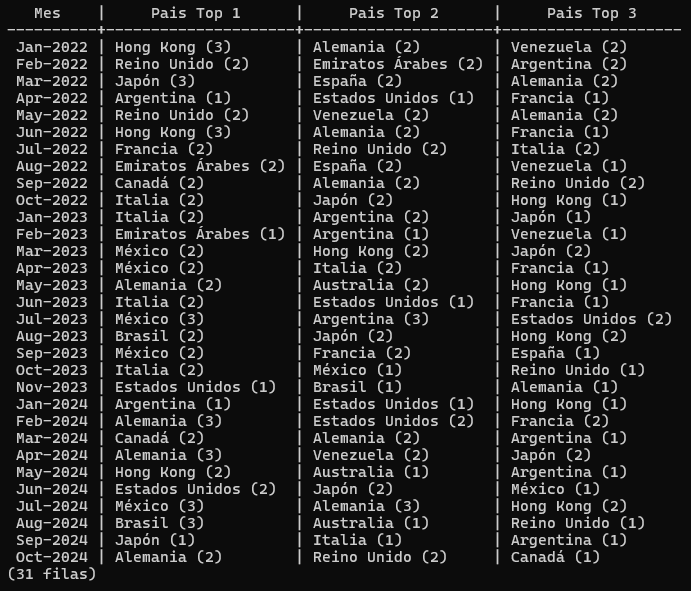
Una manera segura de probar la validez de los insert fue ejecutar una sentencia “SELECT \* FROM tabla;” para cada una de las tablas una vez completada la inserción.

A continuación, se mostrarán las debidas consultas y su resultado:

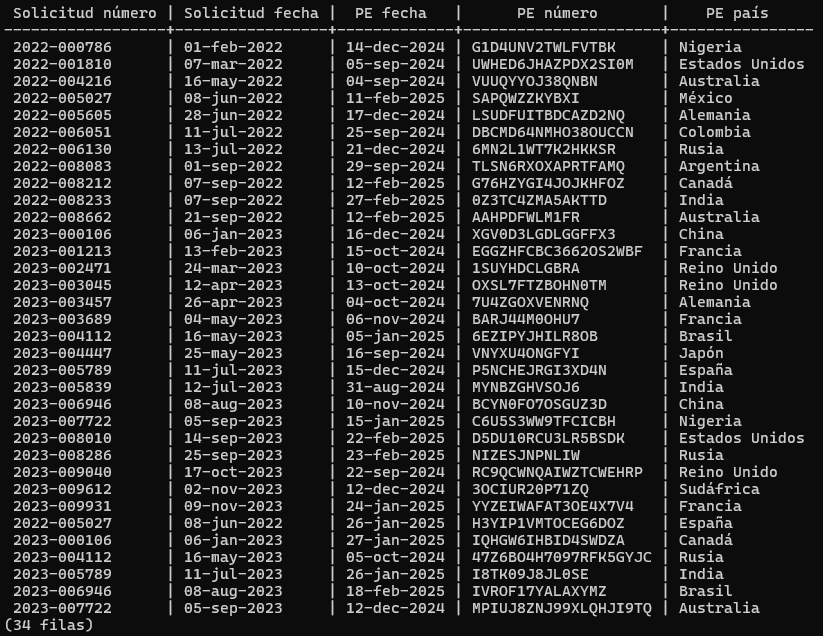
1. Por mes, presentar el total de marcas solicitadas por tipo de signo (denominativo, gráfico, mixto):



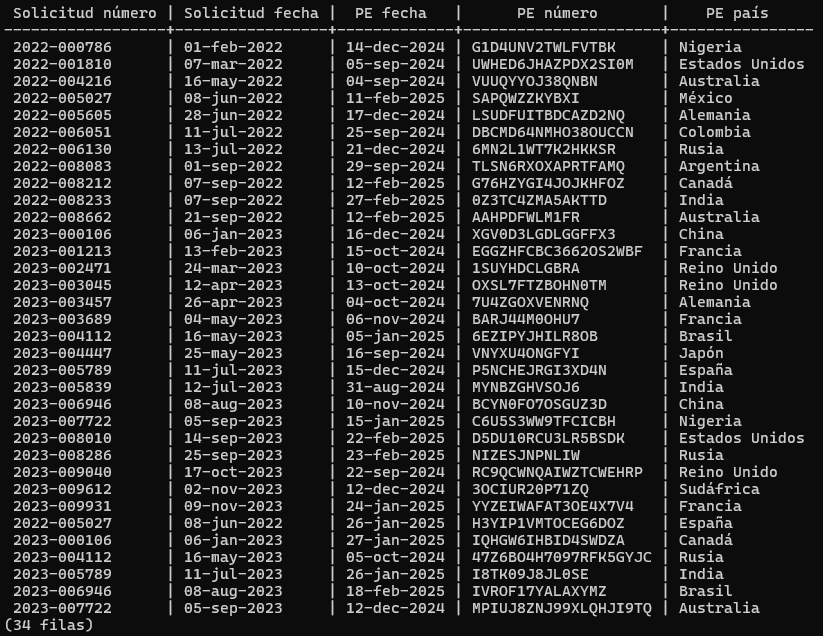
1. Por mes, presentar el top 3 de los países de domicilio de solicitantes con mayor número de solicitudes:



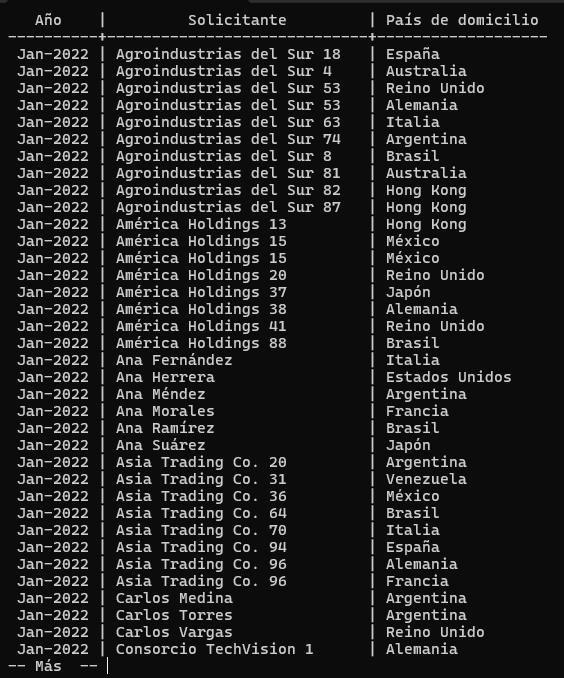
1. Detalles de los números de solicitud con sus prioridades extranjeras (PE):



1. El mismo listado anterior, pero filtrando solamente las filas en donde la fecha de la PE sea mayor o igual que la fecha de la solicitud:



1. Por mes, la lista de solicitantes que no presentaron marcas durante ese mes:



**Decisiones de diseño**

1. Se decidió cambiar la clave primaria de los agentes al número de agente
2. Se agregó “estado” a SOLICITUD

**Lecciones aprendidas**

Por supuesto, el desarrollo de este proyecto no solo nos ayudó a reforzar conceptos relacionados con SQL, sino que también nos acercó al problema muchas veces mencionado de que la población de una base de datos no siempre es una tarea sencilla. Surgieron varios inconvenientes que tuvieron que ser resueltos en equipos; por lo tanto, fue en ejercicio de trabajo en equipo y colaboración.

Más allá de los conceptos aplicados, consideramos que se aprendió principalmente a resolver problemas sobre la marcha y a decidir qué camino tomar para hacerlo más rápido. Claramente, los límites de tiempo suponen un aprendizaje extra de optimización de tiempo a la par que mantener criterios altos de calidad.

**Posibles mejoras o extensiones del proyecto**

Definitivamente, la mejora del archivo de datos de entrada será nuestra primera recomendación a mejorar, para la correcta población de la base de datos se requieren el conjunto completo de datos descritos en la primera especificación del proyecto; no solo una sección de ellos.

También sería positivo no incluir columnas que no aporten información ninguna, tener un archivo de entrada lo más limpio posible es siempre algo positivo.

A nivel de extensión del proyecto, pensamos que se puede alojar una página web que utilice esta base de datos, de modo que los agentes (trabajadores) del SAPI puedan poblar la base de datos a través de un formulario.

También se puede considerar utilizar una red neuronal o un script que, a partir de la imagen de la planilla, sea capaz de rellenar toda la información en cada uno de las tablas. Esto no debería suponer gran reto, siempre que las planillas se mantengan consistentes a lo largo de los años.