(20252Q) 72.37 - Base de Datos I

Grupo 16

Integrantes:

- Hernández, Rodrigo Alejandro - 65522
- Korman, Agustín - 65116
- Nolasco De Carlés, Joaquín - 65368
- Novotny, Nicanor - 65530

Fecha de entrega:

20 / 11 / 2025

Introducción

En este trabajo práctico se desarrolló un sistema de gestión de suscripciones utilizando PostgreSQL, tomando como punto de partida un archivo CSV con la información de pagos realizada por los clientes. El objetivo principal fue automatizar la creación y el mantenimiento de suscripciones a partir de esos pagos, aplicando reglas específicas del negocio como renovaciones dentro de una ventana de 30 días, detección de superposiciones y manejo de períodos de baja.

Para lograrlo fue necesario diseñar las tablas, crear funciones auxiliares, implementar un trigger que genere las suscripciones de forma automática y construir una función de consolidación capaz de reconstruir la línea temporal completa de cada cliente. Además, se documentó el proceso de importación de datos y las dificultades que surgieron durante la implementación.

Asignación de Roles

Para el desarrollo del trabajo, se designaron los siguientes roles:

- Hernandez, Rodrigo Alejandro: Encargado de las funciones.
- Korman, Agustín: Encargado de la investigación y del informe.
- Nolasco De Carlés, Joaquín: Encargado del funcionamiento global del TP.
- Novotny, Nicanor: Encargado del trigger.

Investigación

Para cumplir con los requisitos del TP fue necesario investigar distintos aspectos técnicos de PostgreSQL..

En primer lugar, se profundizó en el manejo de fechas e intervalos dentro de PostgreSQL. Fue necesario investigar cómo funciona internamente INTERVAL '1 month', especialmente en meses con distinta cantidad de días, y cómo afecta el día inicial al resultado del cálculo. También se revisó el comportamiento de la función age(), analizando de qué manera PostgreSQL expresa las diferencias entre fechas en términos de años, meses y días. Esta exploración permitió entender por qué ciertos rangos de fechas no se traducen directamente en una cantidad de meses y cuáles son las particularidades del motor a la hora de realizar estas operaciones con fechas.

Además, los cursores implícitos fueron un tema de investigación importante al desarrollar la función consolidar_cliente . Fue necesario comprender cómo funcionan los bucles del tipo “FOR reg IN SELECT ... LOOP” , cómo se recuperan automáticamente las filas de una consulta y cómo ir acumulando estado entre iteraciones para reconstruir la línea temporal completa de un cliente. Esto permitió identificar períodos continuos, renovaciones y huecos de cobertura de forma ordenada.

Finalmente, también fue necesario investigar cómo trabajar con PostgreSQL de manera local para poder correr los código como superuser. Se exploró cómo inicializar un clúster local, cómo levantar un servidor con pg_ctl, cómo conectarse mediante psql y cómo crear bases de datos y ejecutar scripts SQL de forma aislada. Esta investigación incluyó revisar el funcionamiento de los puertos, logs,y directorios de datos. Comprender este flujo permite tener un entorno local totalmente funcional para así poder ejecutar las instrucciones del TP sin problemas.

Proceso de Importación

Para incorporar la información provista por la cátedra en el archivo pagos.csv, se llevó a cabo un proceso de importación que sigue estrictamente lo solicitado por el enunciado.

Para ejecutar la importación, se utilizó el comando COPY, que permite leer directamente un archivo CSV y convertir cada fila en una tupla válida dentro de la tabla PAGO. Previamente se ejecutaron todos los scripts de creación de tablas, funciones auxiliares y triggers para asegurar que la estructura de la base de datos estuviera completa y que el trigger asociado a la tabla *pago* pudiera operar correctamente durante la carga.

El comando utilizado para realizar la importación fue el siguiente:

```
COPY pago(fecha, medio_pago, id_transaccion, cliente_email, modalidad, monto)
FROM '/[PATH_DEL_ARCHIVO]/pagos.csv'
WITH (
   FORMAT csv,
   HEADER true,
   DELIMITER ','
);
```

Este comando leyó el archivo CSV ubicado en el sistema local y generó una tupla en la tabla *pago* por cada línea del archivo. Al producirse cada inserción, el trigger pagos_before_insert se ejecutó automáticamente, generando las entradas correspondientes en la tabla *suscripción*. De esta manera, la importación no solo cargó correctamente los pagos, sino que también inicializó toda la información de suscripciones requerida para el análisis posterior.

Dificultades Encontradas y Soluciones

Durante la implementación surgieron los siguientes desafíos importantes:

El primer problema que se debió fue simplemente restar fechas (fecha_fin - fecha_inicio) devuelve días, lo cual no es exacto para definir meses comerciales. Usar la función age directamente a veces devolvía “0 años, 0 meses, 30 días” para un mes completo. La solución fue optar por sumar 1 día a la fecha de fin de mes antes de calcular la diferencia. De esta forma, un rango del 1 de enero al 31 de enero se trata como del 1 de enero al 1 de febrero, resultando exactamente 1 mes.

Durante el desarrollo tuvimos un problema conceptual importante con la noción de “suscripción actual”. En la primera versión tomábamos la suscripción actual simplemente como la última por fecha_fin (la de mayor fecha de fin). Eso generó dos efectos indeseados: por un lado, impedía registrar pagos que generaran suscripciones anteriores a la última aunque no se solaparan con ningún período existente; por otro, bloqueaba la posibilidad de renovar suscripciones que no fueran la última, incluso cuando eran las realmente “vigentes” en la fecha del pago. Al releer el enunciado entendimos que cuando se habla de renovar el período “actual” no necesariamente se refiere a la última

suscripción en la línea de tiempo (que puede estar en el futuro), sino a la suscripción que está vigente en la fecha del pago. A partir de eso, ajustamos la lógica: primero identificamos la suscripción que cubre la fecha del pago (si existe) y solo a esa la tratamos como “actual” para aplicar la regla de los 30 días; en caso de no haber ninguna vigente, el pago genera una nueva suscripción siempre que el intervalo resultante no se solape con otros períodos.

Otra dificultad afrontada en el desarrollo del trabajo fue la inyección de claves foráneas en tiempo de ejecución. El archivo CSV no trae una identificación de cada suscripción, pero la tabla PAGO lo refiere como clave foránea. Por ello, se utilizó la cláusula “RETURNING id INTO NEW.suscripcion_id” al momento de insertar en la tabla SUSCRIPCIÓN dentro del trigger. Esto permite vincular el pago con la suscripción recién creada en la misma transacción atómica.

Por último , un obstáculo importante apareció al intentar ejecutar el comando COPY desde el servidor remoto Pampero del ITBA mediante DBVisualizer. Aunque PostgreSQL permite cargar archivos CSV con “COPY FROM”, esta operación requiere permisos elevados como ser superusuario o contar con “pg_read_server_files”. Al nosotros tener cuenta de alumno, no contamos con estos privilegios, por este motivo, cada intento de importar el archivo “pagos.csv” resultaba en un error de acceso denegado. Frente a esta limitación, se decidió realizar la importación en una instancia local de PostgreSQL instalada en la computadora, donde sí se disponía de permisos completos. Una vez creada la base de datos local, se ejecutó COPY sobre la tabla PAGO utilizando el archivo CSV del sistema local, permitiendo que el trigger genere automáticamente las suscripciones correspondientes. De esta forma, la carga de datos pudo completarse utilizando COPY y respetando la consigna..

Conclusión

Para concluir , este trabajo permitió integrar varios conceptos de PostgreSQL en un caso práctico que combina reglas de negocio, validación de datos y automatización mediante triggers. A partir del archivo de pagos se logró construir un sistema capaz de generar y analizar suscripciones de manera coherente, detectando renovaciones, períodos de baja y posibles inconsistencias. También surgieron desafíos reales como el manejo correcto de fechas o las limitaciones del comando “COPY” en el servidor remoto que aportaron una experiencia más completa al proceso. En conjunto, el desarrollo permitió entender mejor cómo modelar y resolver problemas típicos de administración de datos en entornos reales.