Proyecto Final del Curso de Arquitectura de Bases de Datos

Título: "Global Stream Hub: Arquitectura de Base de Datos para Plataforma de Streaming Multimedia"

Fecha de Presentación: Jueves, 18 de Diciembre de 2025, 6:00 PM.

---

1. Visión General del Proyecto

El objetivo de este proyecto es diseñar, implementar y documentar una arquitectura de base de datos robusta y escalable en PostgreSQL para soportar una plataforma global de streaming de video y música. La base de datos servirá como el componente central de datos para el ecosistema de la plataforma, alimentando las capas de backend, APIs, y potencialmente sistemas de análisis y publicidad. Se enfatizará la aplicación de principios arquitectónicos modernos como la arquitectura Medallón, la auditoría de datos, el manejo de historiales y la preparación para la exposición de datos a través de APIs.

---

2. Escenario del Negocio: Global Stream Hub

Global Stream Hub es una nueva plataforma de entretenimiento que ofrece contenido de video y música a suscriptores a nivel mundial. La empresa opera bajo un modelo de negocio híbrido:

- **Suscripción Premium:** Usuarios pagan una tarifa mensual para acceso sin publicidad y contenido exclusivo.
- **Modelo bajo costo con Publicidad:** Usuarios con este tipo de suscripción acceden al contenido con anuncios intercalados.

La plataforma debe manejar una gran cantidad de datos y transacciones concurrentes, incluyendo:

- Gestión de usuarios y sus perfiles (suscripción, preferencias).
- Catálogo de contenido multimedia (videos, canciones, películas, series), organizado jerárquicamente por categorías y subcategorías.
- Reproducción de contenido y seguimiento de actividad del usuario.
- Gestión de listas de reproducción y contenido favorito.
- Sistema de publicidad (gestión de campañas, despliegue de anuncios, seguimiento de impresiones, a partir del contenido (categorías) que la publicidad sea una o otra).

---

3. Requerimientos Arquitectónicos y de Base de Datos

Los estudiantes deben diseñar e implementar una solución utilizando **PostgreSQL (versión 17)** en Azure como servicio, con DBeaver como herramienta de interacción.

3.1. Arquitectura Medallón

Se debe implementar una arquitectura Medallón para organizar los datos en distintas capas de refinamiento:

- **Capa Bronze (Ingesta Cruda):**
  - **Propósito:** Almacenar datos en su formato original, sin procesar, tal como llegan de las fuentes. Inmutable.
  - **Fuentes de Datos (Simuladas/Estructuradas):**
    - `user_registrations.csv`: Datos de nuevos usuarios (ID, nombre, email, fecha de registro, país, tipo de suscripción).
    - `raw_streaming_logs.json` (o `streaming_logs.csv` si el JSON es complejo): Logs de actividad de reproducción (user_id, content_id, timestamp_inicio, timestamp_fin, dispositivo, tiempo_visto_pct).
    - `raw_catalog_data.csv`: Información del catálogo multimedia (content_id, título, tipo [video/music], género, director/artista, año, descripción).
    - `raw_ad_impressions.log`: Logs de despliegue de anuncios (timestamp, ad_id, user_id, content_id, ad_placement).
  - **Implementación:** Tablas en el schema `bronze` que reflejen la estructura de los archivos de entrada.
- **Capa Silver (Datos Limpios y Enriquecidos):**
  - **Propósito:** Datos de la capa Bronze que han sido limpiados, validados, des-duplicados y enriquecidos (ej: uniendo logs con información de usuario, parseando metadatos).
  - **Contenido:** Tablas con tipos de datos correctos, esquemas normalizados, datos enriquecidos (ej: calculando duración de reproducción).
  - **Implementación:** Tablas en el schema `silver` con estructuras optimizadas y tipos de datos adecuados.
- **Capa Gold (Datos Curados y Agregados):**
  - **Propósito:** Datos de la capa Silver optimizados para análisis de negocio, BI y para ser expuestos vía APIs. Típicamente con modelos dimensionales (estrellas o copos de nieve) o agregaciones.
  - **Contenido:** Tablas de hechos y dimensiones, agregaciones de métricas clave (ej: horas vistas por usuario/mes, popularidad de contenido por categoría, rendimiento de anuncios).
  - **Implementación:** Tablas de hechos y dimensiones en el schema `gold`, y el uso de Vistas Materializadas para métricas de alto rendimiento.

3.2. Tablas y Estructura de Datos (Aprox. 15-25 Tablas)

Se espera un diseño de base de datos relacional bien normalizado en la capa Silver y dimensional en la capa Gold.

- **Ejemplos de Tablas:**
  - `sales.customers` (o `users.profiles`)
  - `sales.products` (o `catalog.media_content`)
  - `sales.orders` (o `streaming_sessions`)
  - `sales.order_items` (o `streaming_playbacks`)
  - `sales.inventory` (o `content_availability`)
  - Tablas de `streaming_logs` (procesados)
  - Tablas de `ad_performance` (procesadas)
  - Tablas de `subscriptions`
  - Dimensiones para `users`, `content`, `devices`, `genres`, `regions`, `dates`, `time_of_day`.
  - Tablas de hechos para `user_activity_summary`, `ad_impressions_summary`, `content_popularity`.

3.3. Auditoría y Seguridad (RNF-05, RNF-06)

- **Pistas de Auditoría:** Implementar auditoría interna en tablas maestras (ej: `created_at`, `updated_at`, `updated_by`) y un sistema centralizado de logs de operaciones (`audit.operation_log`).
- **Clasificación y Sensibilidad de Datos:**
  - Identificar datos sensibles (ej: información personal de usuario, historial de visualización, datos de pago).
  - Proponer políticas de acceso y enmascaramiento de datos (conceptual).
  - Diseñar schemas y roles de acceso en PostgreSQL (`etl_process_role`, `bi_analyst_role`, `api_consumer_role`) para restringir el acceso según el principio de mínimo privilegio.
- **Log Histórico de Transacciones:**
  - Implementar tablas de historial (`history.table_name`) y triggers `AFTER INSERT OR UPDATE OR DELETE` para capturar el estado de las filas antes y después de los cambios.
  - Utilizar columnas `JSONB` en las tablas de historial para almacenar las versiones `OLD` y `NEW` de las filas.

3.4. Particionamiento de Tablas (opcional)

- **Propósito:** Optimizar el rendimiento de consultas y la gestión de datos para tablas muy grandes (logs de streaming, historial de reproducción, impresiones de anuncios).
- **Estrategia:** Particionar tablas clave por rangos de tiempo (ej: `streaming_logs` particionada por mes o año, `ad_impressions` por día).

3.5. Vistas y Vistas Materializadas

- **Vistas Normales:** Crear vistas para simplificar consultas complejas de negocio, como la visualización del historial de reproducción de un usuario, listas de contenido por género, o el estado de suscripción de un usuario.

  Las vistas mínimas a implementar deben ser, sin embargo el estudiantes podra implementar mas vistas sin problema:

  - **`vw_user_profile_details`:** Proporcionar información completa del perfil de un usuario, incluyendo su tipo de suscripción y la fecha de la última actividad registrada.
  - **`vw_content_catalog_summary`:** Ofrecer un resumen del catálogo de contenido, incluyendo título, tipo, género principal y popularidad (basada en conteos de reproducción).
  - **`vw_user_playback_history`:** Mostrar el historial de reproducciones de un usuario específico, incluyendo detalles del contenido y la duración de la sesión.
  - **`vw_playlist_details`:** Mostrar el contenido de una lista de reproducción creada por un usuario, incluyendo detalles de cada ítem en la lista.
  - **`vw_ad_performance_summary`:** Agrupar el rendimiento de los anuncios (impresiones, clics, tiempo de despliegue) por campaña publicitaria o tipo de contenido.
- **Vistas Materializadas:** Diseñar Vistas Materializadas para métricas de alto rendimiento requeridas por APIs analíticas o dashboards:
  - Ej: `mv_monthly_user_activity` (horas vistas por usuario al mes).
  - Ej: `mv_content_popularity_by_genre` (conteos de reproducciones por contenido y género).
  - Ej: `mv_ad_performance_metrics` (impresiones, clics, CTR por campaña publicitaria).

---

4. Entregables del Proyecto Final

1. **Diagrama de Arquitectura de la Solución Tecnológica: (El profesor los suministrara)**
   - Diagrama visual que muestre las capas (Bronze, Silver, Gold), las bases de datos principales (PostgreSQL), la simulación de fuentes de datos. Incluir

componentes conceptuales como Frontend, Backend, Kubernetes, Message Queues (si se mencionan).

2. **Diagramas del Modelo de Datos (ERD):**
    - Diagramas separados para cada schema (Bronze, Silver, Gold), mostrando tablas, columnas, tipos de datos, relaciones (claves foráneas), índices y particionamiento. (Los comentarios no deberán ser mostrador en el diagrama, sin embargo si es un elemento relevante del diseño de la bases de datos)
3. **Scripts SQL Completos:**
    - Un archivo `.sql` o un conjunto de scripts bien organizados y documentados, que creen toda la estructura de la base de datos (schemas, tablas, secuencias, índices, triggers, funciones, vistas, vistas materializadas, roler, permisos, etc). Deben ser ejecutables en orden.
4. **Documento de Diseño y Justificación (PDF/Markdown):**
    - Introducción al problema de negocio.
    - Descripción detallada de la arquitectura elegida (incluyendo el diagrama).
    - Explicación del diseño de la base de datos: justificación de la arquitectura Medallón, decisiones de modelado (normalización vs. dimensional), elección de tipos de datos (incluyendo `JSONB` y `SERIAL`/secuencias).
    - Detalle de las estrategias de particionamiento implementadas (sise desarrolla el particionamiento el cual se seteo como opcional).
    - Explicación de los mecanismos de auditoría y log histórico (triggers, tablas de historial, `audit.operation_log`).
    - Justificación de las vistas y vistas materializadas creadas, y cómo sirven a las APIs.
    - Consideraciones de seguridad (roles, permisos).
    - Conclusiones y posibles mejoras futuras.
5. **Presentación Magistral (Oral):**
    - Exposición clara y concisa del proyecto (máximo 20 minutos).
    - Demostración del funcionamiento de la base de datos (ejecución de consultas sobre vistas y MVs).
    - Capacidad para explicar la arquitectura, las decisiones de diseño y cómo la base de datos soporta los requerimientos del negocio.
6. **Entrega de archivo .ZIP con todos los entregables:**
    - Todo el código fuente (scripts SQL), diagramas, documento de diseño y cualquier otro archivo relevante organizado en un repositorio público.