Inserción, Hashing e Índices Incrementales con Sockets

Proyecto `hash_table`: CSV + `tracks.idx` + `nameidx`

Documentación técnica

30 de octubre de 2025

1. Resumen

Implementamos una tubería completa para **agregar tracks** al dataset y hacerlos **buscables al instante** por *ID* y por *nombre/artista*:

- El cliente envía `ADD|...` al servidor por **socket TCP**.
- El servidor añade la fila al **CSV** (append), actualiza el índice por **ID** (`tracks.idx`) y registra un **delta incremental** para el índice de texto en `nameidx/updates/`.
- El programa `p1-dataProgram` busca por **palabras** fusionando **base** + **delta** y muestra los **más recientes primero**.
- Se añadieron *fallbacks* para filas *cortas* y mejoras de impresión/lookup.

2. Flujo de inserción (`ADD`)

Protocolo del cliente

```
ADD|<track_id>|<name>|<artist>|<album>|<duration_ms>\n
```

Pasos en el servidor (`track_server`)

1. **Parseo**: separa por "|" y arma `TrackRecord`.
2. **Append al CSV**: abre `merged_data.csv` en modo append, bloquea, escribe una fila *corta* de 5 campos, agrega `\n` y obtiene el **offset** de inicio de línea (posición en bytes).
3. **Índice por ID** (`tracks.idx`):
   - Calcula h = FNV-1a (64 bits) del `track_id`.
   - Inserta en tabla hash con *linear probing*. Cada slot es `{hash, offset}`.
   - Ante colisión con mismo `hash`, **verifica** el `track_id` real leyendo el CSV en el `offset`.
4. **Índice de texto incremental** (`nameidx/updates/`):
   - Normaliza `name` y `artist` (minúsculas, tildes básicas fuera, ñ→n), tokeniza.
   - Para cada token: h_{tok} = FNV-1a, bucket $b = h_{\text{tok}}$ & $0x$FF.
   - Anexa línea en `nameidx/updates/b%02x.log`:

     ```
     <hash_token_hex16> <offset_csv_decimal>
     ```
5. **Respuesta**: `OK <offset>` o `ERR <mensaje>`.

3. Estructura de `tracks.idx` y hashing

Header y slots

- Header `IdxHeader`: `magic=ÏDX1TRK"`, `capacity` (potencia de 2), `key_col` (columna del `track_id` en el CSV), etc.

- **Slots** contiguos: cada uno es `{uint64_t hash, uint64_t offset}`.

Hash FNV-1a (64 bits)

- Rápido y estable; buen *spread* sobre claves de texto.
- Si el resultado fuese 0, lo forzamos a 1 para evitar el valor "vacío".

Inserción y lookup

- Posición inicial: $i = \text{hash} \;\&\; (\text{capacity} - 1)$.
- *Linear probing*: avanza circularmente hasta slot vacío o `hash` igual.
- Verificación por **contenido**: compara `track_id` del CSV en el offset; así evitamos falsos positivos por colisiones.

4. Índice de nombres: base + delta

Base (`nameidx/bXX.idx`)

Archivos binarios por bucket ($b = 0 \ldots 255$) con bloques:

$$[\text{hash}][\text{df}][\text{pad}]\underbrace{[\text{offset}_1, \ldots, \text{offset}_{\text{df}}]}_{\text{postings ordenadas}}$$

Se generan con `build_name_index` recorriendo todo el CSV.

Delta incremental (`nameidx/updates/bXX.log`)

- Se escribe **en caliente** por el servidor en cada alta.
- Formato texto, una línea por token: hash (hex) y offset (decimal).
- Permite **ver** las altas nuevas sin reconstruir la base.

Búsqueda por palabras en `p1-dataProgram`

Para cada palabra introducida por el usuario:

1. Normaliza y tokeniza; usa el **primer token** de cada palabra.
2. Carga la **base** (`bXX.idx`) y el **delta** (`updates/bXX.log`); ordena y hace *unique* el delta.
3. **Fusiona** base+delta (ambas listas ordenadas) en una única lista ordenada.
4. Si hay varias palabras, hace **intersección** de listas (operador AND).
5. Lee cada `offset` del CSV y **imprime** una línea compacta.
6. **Recientes primero**: muestra los últimos `MAX_SHOW` offsets.

5. Filas "cortas" vs. CSV completo

Las nuevas inserciones escriben 5 campos: `track_id`, `name`, `artist`, `album`, `duration_ms`. El dataset original tiene más columnas. Para que todo funcione sin reconstruir:

- **Impresión**: si la línea tiene exactamente 5 campos, usamos posiciones 0–2 para (`id`, `name`, `artist`) como *fallback*.
- **Lookup por ID**: si `key_col` queda fuera de rango en una fila corta, comparamos con la columna 0.

Si se desea *homogeneidad total*, puede escribirse una fila *larga* con todas las columnas del header rellenando vacíos con `""`.

6. Comunicación por socket (TCP)

Servidor

```
socket -> bind -> listen -> accept -> (leer una linea) -> responder -> close
```

Mono-hilo, un comando por conexión, suficiente para la práctica/entrega.

Cliente

```
./track_client 127.0.0.1 5555 ADD feid-006 "FERXXO 151" "Feid" \
  "Mor, No Le Temas a la Oscuridad" 185000
```

Equivalente con `nc`:

```
printf 'ADD|feid-007|Yandel 150 (Remix)|Feid|Single|197000\n' | nc 127.0.0.1 5555
```

7. Comandos de prueba

Arranque del servidor

```
./track_server merged_data.csv tracks.idx nameidx 5555
# track_server escuchando en puerto 5555 (CSV=... IDX=... NAMEIDX=nameidx)
```

Altas por socket

```
./track_client 127.0.0.1 5555 ADD feid-005 "Feliz Cumpleaos Ferxxo" "Feid" \
  "Te Pirateamos el lbum" 204000
```

Verificación de deltas

```
ls -l nameidx/updates
tail -n 5 nameidx/updates/b*.log
```

Búsqueda en `p1-dataProgram`

```
./p1-dataProgram
# Palabra #1: feid
# 3) Realizar bsqueda --> vers arriba lo recin insertado
```

8. Consideraciones y mejoras

- **Paginación** o toggle "recientes/antiguos" en la UI.
- **Duplicados por `track_id`**: rechazar si ya existe la clave.
- **Comandos** de servidor: `PING`, `LOOKUP|<id>`, etc.
- **Filas completas**: escribir todas las columnas del header.
- **Compactación** del delta: job periódico que mergee `updates` a la base.

9. Conclusión

La arquitectura combina **append-only** al CSV, **hashing** con verificación de contenido para el índice por ID y un **delta incremental** para el índice invertido de texto. Así, las altas quedan **disponibles al instante** para búsquedas por nombre/artista sin reconstrucciones costosas, manteniendo el rendimiento y la simplicidad del formato en disco.