Inserción, Hashing, Índices Incrementales y Búsqueda por Socket

Proyecto `hash_table`: CSV + `tracks.idx` + `nameidx` (+ `SEARCH`)

Documentación técnica

30 de octubre de 2025

1. Resumen

Construimos una solución completa para:

- **Agregar** registros al dataset (CSV) vía `ADD` por socket.
- Actualizar índices: **ID** (`tracks.idx`) y **texto** incremental (`nameidx/updates`).
- **Buscar** por *ID* (en app local) y por *nombre/artista* desde el **cliente** usando el nuevo comando `SEARCH`.
- Mostrar resultados **recientes primero** (últimos offsets).

2. Arquitectura general

- **CSV** (`merged_data.csv`): almacenamiento principal (*append-only*).
- **tracks.idx**: tabla hash con open addressing (*linear probing*) para lookup por `track_id`.
- **nameidx**: índice invertido por tokens:
  - *Base* binaria: `nameidx/bXX.idx` (ordenada).
  - *Delta incremental*: `nameidx/updates/bXX.log` (texto).
- **Servidor TCP** (`track_server`): recibe `ADD`/`SEARCH`, escribe/consulta y responde.
- **Cliente TCP** (`track_client`): envía comandos de forma simple.
- **App local** (`p1-dataProgram`): búsqueda por ID y palabras (base+delta), recientes primero.

3. Flujo de inserción `ADD`

Protocolo (cliente → servidor)

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

Servidor (`track_server`)

1. **Parseo** y armado de `TrackRecord`.
2. **Append al CSV**: escribe fila corta (5 campos) y obtiene *offset* de inicio de línea.
3. **Índice por ID** (`tracks.idx`):
   - Hash FNV-1a (64-bit) sobre `track_id`.
   - Inserción con *linear probing*: guarda `{hash, offset}`.
   - *Verificación por contenido*: abre CSV en el offset, confirma `track_id` exacto.
4. **Índice de texto (delta)**: normaliza (*lowercase*, sin tildes básicas, ñ→n), tokeniza `name` y `artist`, y por cada token:

   ```
   <hash_token_hex16> <offset_csv_decimal>\n --> nameidx/updates/bXX.log
   ```

5. **Respuesta**: `OK <offset>` ó `ERR <detalle>`.

4. Índice por ID: `tracks.idx`

- Header `IdxHeader`: `magic="IDX1TRK"`, `capacity` (potencia de 2), `key_col`, etc.
- Slots contiguos: `{uint64_t hash, uint64_t offset}`.
- Posición inicial: $i = \text{hash} \;\&\; (\text{capacity} - 1)$.
- Colisiones: *linear probing* con verificación leyendo el CSV.

5. Índice de texto: base + delta

Base (`nameidx/bXX.idx`)

Bloques binarios:

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

Se genera con `build_name_index` recorriendo el CSV completo.

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

- Escrito por el servidor en cada `ADD`.
- Permite *visibilidad inmediata* sin reconstruir la base.

Fusión y búsqueda por palabras

Para cada palabra:

1. Normalizar y tokenizar; se usa el *primer token*.
2. Cargar **base** y **delta**, ordenar/unique el delta.
3. **Fusionar** (merge) listas ordenadas; si hay varias palabras, **intersección** (AND).
4. Abrir CSV por offset y **mostrar** línea compacta.
5. Mostrar **recientes primero**: últimos `MAX_SHOW`.

6. Nueva función por socket: `SEARCH`

Protocolo

```
SEARCH|<palabra1>[|<palabra2>][|<palabra3>]\n
```

Respuesta del servidor:

```
OK <N>
<track_id> | <track_name> | <artist> | <date> | <region>
...
END
```

donde `N` es el número de filas que se van a devolver (máximo los últimos `MAX_SHOW`, recientes primero).

Lado servidor

1. Parsear palabras, normalizarlas y tokenizarlas (primer token).
2. Por token: cargar postings *base* y *delta*, fusionar.
3. Para múltiples palabras: intersección ordenada (AND).
4. Abrir CSV y emitir las últimas `MAX_SHOW` líneas compactas.

Lado cliente

```
./track_client 127.0.0.1 5555 SEARCH feid
./track_client 127.0.0.1 5555 SEARCH feid 151 # AND
```

El cliente imprime la respuesta tal cual: `OK N`, filas y `END`.

7. Filas "cortas" y compatibilidad

Las altas escriben 5 campos: `track_id`, `name`, `artist`, `album`, `duration_ms`. Para mantener compatibilidad:

- **Impresión compacta**: si hay exactamente 5 campos, se toman 0..2 como `id`, `name`, `artist`.
- **Lookup por ID**: si `key_col` no existe en una fila corta, se compara contra la columna 0 (`track_id`).

Si se desea homogeneidad total, puede escribirse la fila completa con todas las columnas del header (rellenando vacíos con `""`).

8. Comunicación TCP

Servidor

```
socket -> bind -> listen -> accept -> leer comando -> responder -> close
```

Comandos soportados: `ADD` y `SEARCH` (extensible a `PING`, `LOOKUP`, etc.).

Cliente

```
./track_client <host> <port> ADD <track_id> <name> <artist> <album> <duration_ms>
./track_client <host> <port> SEARCH <pal1> [<pal2>] [<pal3>]
```

9. Pruebas rápidas

Servidor

```
./track_server merged_data.csv tracks.idx nameidx 5555
```

Altas

```
./track_client 127.0.0.1 5555 ADD feid-201 "Castigo" "Feid" "Single" 184000
```

Búsqueda por nombre/artista

```
./track_client 127.0.0.1 5555 SEARCH feid
./track_client 127.0.0.1 5555 SEARCH feid 151
```

Verificación del delta

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

10. Consideraciones y mejoras

- **Paginación** (`SEARCH` con `--page`/`--limit`).
- **Duplicados** por `track_id`: rechazo en `ADD`.
- **Comandos** extra: `PING`, `LOOKUP`, `STATS`.
- **Compactación** del delta: *merge* periódico de `updates` hacia `bXX.idx`.

11. Conclusión

La solución combina un *append-only* CSV, hashing verificado por contenido para `track_id` y un índice invertido *base+delta* para texto. Con el nuevo `SEARCH` por socket, las altas quedan **buscables al instante** desde clientes remotos, manteniendo buen rendimiento y simplicidad en disco.