

Backend de Aplicaciones

Consigna – Pre-enunciado Recuperatorio 2025

Contexto: base de datos de aplicación de comercialización de música.

Objetivo Preparar una **aplicación Java (consola)** que deje lista **la base de datos H2 en memoria, el mapeo JPA/Hibernate**, y una **validación mínima** sobre la estructura y los datos. Todo apuntado a que, en el día del parcial, solo tengas que agregar la **carga de los datos de un CSV** y los **procesos específicos** que se soliciten.

En otras palabras: **antes del parcial** dejar listo **todo el andamiaje técnico** (BD H2, DDL, JPA, secuencias, relaciones) para la **BD completa**. **El día del parcial** se entregará un **bloque de datos (subset)** sobre un **segmento de la BD**, de complejidad y estructura similar a la usada en el parcial, para llevar a cabo las consignas del examen.

1) Requisitos técnicos (obligatorios)

- **Java 17 o superior, Maven.**
- Librerías: **Lombok, JDBC, JPA/Hibernate.**
- **H2 en memoria** (embedded) obligatorio.
- **DDL:** usar el archivo provisto (o construido por vos) **database-ddl.sql** con la estructura de tablas y secuencias para la base de datos adaptada a H2.
- **Convenciones de nombres:**
 - Tablas y columnas en **MAYÚSCULAS** con **SNAKE_CASE** (viene así en el DDL).
 - Clases Java en **UpperCamelCase**; campos Java en **lowerCamelCase**.
 - Mapear nombres distintos con **@Column(name = "...")**.
- **Secuencias** en BD:
 - Cada tabla con clave primaria numérica utiliza una **secuencia propia**.
 - En el DDL se utiliza el patrón: **ID_XYZ INTEGER NOT NULL DEFAULT NEXT VALUE FOR SEQ_XYZ_ID**

2) Estructura de proyecto (SUGERIDA)

El alumno puede optar por la estructura que prefiera y con la que se sienta más confiado para la realización del parcial. La que sigue es **solamente una guía** alineada con los pasos que venimos usando en JDBC y JPA.

```
/ (root)
└── src/
    └── main/
        └── java/utnfc/isi/back/...
            └── catedra/
                └── infra/
                    └── DataSourceProvider.java
```

← paquetes según criterio de proveedor JDBC (H2)

```

en memoria)
|   |   |   | LocalEntityManagerProvider.java      ← proveedor de
EntityManager (JPA)
|   |   |   | DbInitializer.java                 ← ejecuta database-
ddl.sql vía JDBC
|   |   |   | domain/                         ← entidades JPA
|   |   |   | (modelo)
|   |   |   |   | Artist.java
|   |   |   |   | Album.java
|   |   |   |   | Employee.java
|   |   |   |   | Customer.java
|   |   |   |   | Genre.java
|   |   |   |   | MediaType.java
|   |   |   |   | Playlist.java
|   |   |   |   | Track.java
|   |   |   |   | Invoice.java
|   |   |   |   | InvoiceItem.java
|   |   |   |   | PlaylistTrack.java
|   |   |   |   | repo/                         ← opcional (repos JPA
o EM directo)
|   |   |   | resources/
|   |   |   |   | META-INF/
|   |   |   |   |   | persistence.xml          ← unidad de
persistencia JPA
|   |   |   |   |   | sql/
|   |   |   |   |   |   | database-ddl.sql      ← DDL H2 + secuencias
|   |   |   |   |   | App.java                  ← Main: orquesta init
+ validación
└ pom.xml

```

3) Tareas a realizar

3.1 Inicialización de BD

Implementar **DbInitializer** o algún esquema de inicialización que:

- Abra una conexión H2 **en memoria** (ejemplo de URL:
`jdbc:h2:mem:database;DB_CLOSE_DELAY=-1`)
- Ejecute **completo database-ddl.sql** (desde el *classpath*), incluyendo:
 - **creación de secuencias** (`CREATE SEQUENCE ...`),
 - **creación de tablas** (todas las tablas),
 - **definición de claves primarias y foráneas**,
 - **creación de índices**.
- Cierre recursos correctamente (usar preferentemente `try-with-resources`).

3.1.1 Estructura de la base de datos

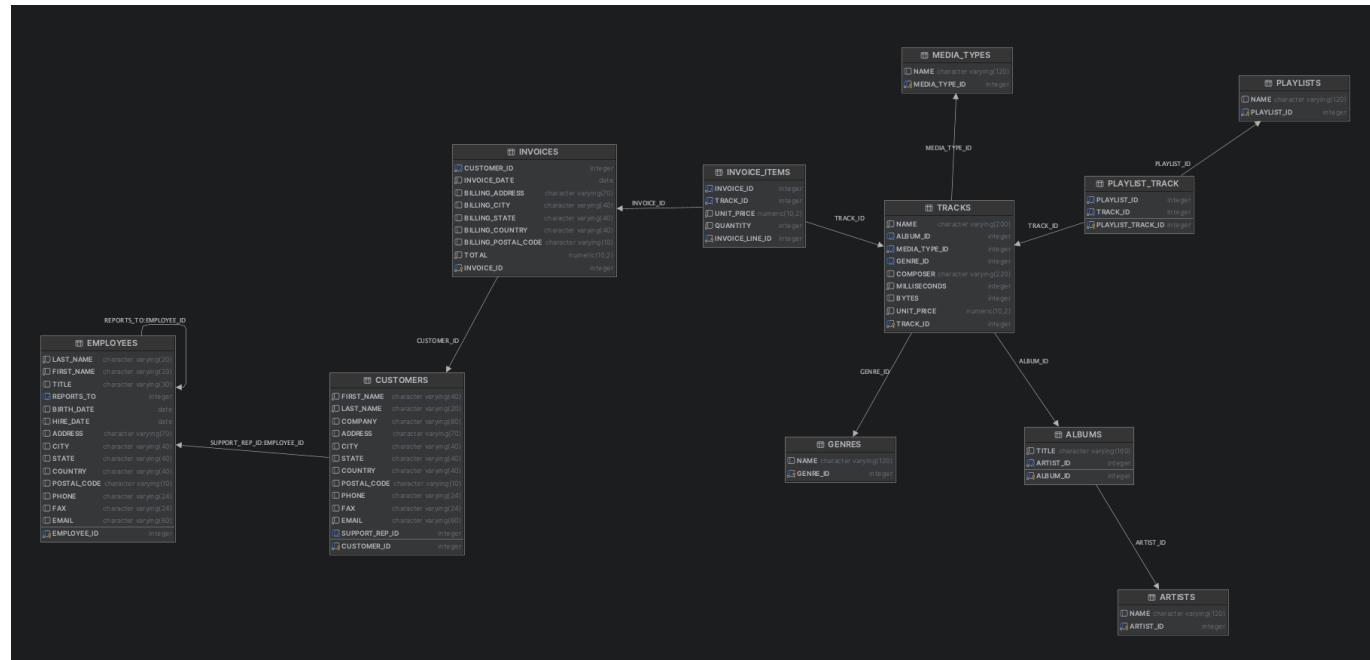
La base adaptada contiene, entre otras, las siguientes tablas principales:

- **ARTISTS** (ARTIST_ID, NAME)
- **ALBUMS** (ALBUM_ID, TITLE, ARTIST_ID)

- **EMPLOYEES** (EMPLOYEE_ID, LAST_NAME, FIRST_NAME, REPORTS_TO, BIRTH_DATE, HIRE_DATE, ...)
- **CUSTOMERS** (CUSTOMER_ID, FIRST_NAME, LAST_NAME, EMAIL, SUPPORT_REP_ID, ...)
- **GENRES** (GENRE_ID, NAME)
- **MEDIA_TYPES** (MEDIA_TYPE_ID, NAME)
- **PLAYLISTS** (PLAYLIST_ID, NAME)
- **TRACKS** (TRACK_ID, NAME, ALBUM_ID, MEDIA_TYPE_ID, GENRE_ID, COMPOSER, MILLISECONDS, UNIT_PRICE, ...)
- **INVOICES** (INVOICE_ID, CUSTOMER_ID, INVOICE_DATE, BILLING_ADDRESS, TOTAL, ...)
- **INVOICE_ITEMS** (INVOICE_LINE_ID, INVOICE_ID, TRACK_ID, UNIT_PRICE, QUANTITY)
- **PLAYLIST_TRACK** (PLAYLIST_TRACK_ID, PLAYLIST_ID, TRACK_ID)

DER

A continuación se agrega un **DER simplificado** de la base para ayudar con la comprensión de la estructura.



3.2 Capa de infraestructura

- **DataSourceProvider:**
 - Expone un `javax.sql.DataSource` (H2 en memoria) para **JDBC** y **JPA**.
 - Centraliza la URL, usuario y contraseña (si aplica).
- **LocalEntityManagerProvider:**
 - Configura la `EntityManagerFactory` con Hibernate y H2.
 - Debe usar `hibernate.hbm2ddl.auto=none` (porque el DDL lo ejecuta `DbInitializer`).
 - Puede habilitar `show_sql` y `format_sql` para debug (opcional).
 - Usa el **mismo datasource** que `DataSourceProvider`.

Importante: mantener nombres coherentes con los materiales de referencia, ya que el día del parcial se reutilizarán para inicializar y validar la estructura antes de resolver las consignas.

3.3 Entidades JPA (mapeos)

Crear entidades para **todas** las tablas principales del DDL (nombres sugeridos):

- Artist → ARTISTS (ARTIST_ID, NAME)
- Album → ALBUMS (ALBUM_ID, TITLE, ARTIST_ID)
- Employee → EMPLOYEES (EMPLOYEE_ID, LAST_NAME, FIRST_NAME, REPORTS_TO, ...)
- Customer → CUSTOMERS (CUSTOMER_ID, FIRST_NAME, LAST_NAME, EMAIL, SUPPORT_REP_ID, ...)
- Genre → GENRES (GENRE_ID, NAME)
- MediaType → MEDIA_TYPES (MEDIA_TYPE_ID, NAME)
- Playlist → PLAYLISTS (PLAYLIST_ID, NAME)
- Track → TRACKS (TRACK_ID, NAME, ALBUM_ID, MEDIA_TYPE_ID, GENRE_ID, COMPOSER, MILLISECONDS, ...)
- Invoice → INVOICES (INVOICE_ID, CUSTOMER_ID, INVOICE_DATE, TOTAL, ...)
- InvoiceItem → INVOICE_ITEMS (INVOICE_LINE_ID, INVOICE_ID, TRACK_ID, UNIT_PRICE, QUANTITY)
- PlaylistTrack → PLAYLIST_TRACK (PLAYLIST_TRACK_ID, PLAYLIST_ID, TRACK_ID)

3.3.1 Indicaciones específicas:

- Definir **@Id** y la estrategia de generación acorde a las **secuencias** definidas en el DDL. Ejemplo (orientativo):

```

@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE, generator =
"artist_seq")
@SequenceGenerator(name = "artist_seq", sequenceName =
"SEQ_ARTIST_ID", allocationSize = 1)
private Integer idArtist;

```

- Relaciones sugeridas:
 - Album – Artist: **@ManyToOne(optional = false)** desde **Album** a **Artist**.
 - Track – Album: **@ManyToOne(optional = true)** (algunos tracks pueden no tener album).
 - Track – MediaType: **@ManyToOne(optional = false)**.
 - Track – Genre: **@ManyToOne(optional = true)**.
 - Customer – Employee (SupportRep): **@ManyToOne(optional = true)**.
 - Invoice – Customer: **@ManyToOne(optional = false)**.
 - InvoiceItem – Invoice: **@ManyToOne(optional = false)**.
 - InvoiceItem – Track: **@ManyToOne(optional = false)**.
 - PlaylistTrack – Playlist y Track: **@ManyToOne(optional = false)** a cada uno.
- Usar **@Column(name = "...")** si el nombre de la columna no coincide con el nombre del atributo Java.
- Las tablas **ALBUMS**, **PLAYLIST_TRACK** y **INVOICE_ITEMS** son evidentemente relaciones muchos a muchos, sin embargo no se espera el mapeo many to many sino que se contempla el mapeo many to

one desde la relación que corresponda a cada una de sus relaciones referenciadas con eso va a ser suficiente el día del parcial.

3.4 Repositorios (opcional)

Se puede implementar **repos JPA** simples o usar **EntityManager/TypedQuery** directamente desde la capa de aplicación.

Sugerencia de repos opcionales:

- **ArtistRepository**
- **AlbumRepository**
- **TrackRepository**
- **InvoiceRepository**

Con métodos básicos como **findById**, **findAll**, de acuerdo a lo documentado en el material de JPA y alguna consulta de ejemplo (**findByNameContaining**, etc.).

No es obligatorio entregar repositorios en esta instancia, pero podría facilitar el trabajo el día del parcial.

3.5 Main **App.java** (Sugerido)

Se sugiere la construcción de una clase con un método **main** que compruebe al menos lo siguiente:

1. Ejecutar **DbInitializer** y mostrar **si la inicialización fue exitosa**.
2. Construir un **EntityManager** vía **LocalEntityManagerProvider**.
3. Realizar un **smoke test** mínimo, por ejemplo:
 - Contar registros en **ARTISTS** o **GENRES** (si se genera algún seed de prueba), o
 - Crear y persistir un **Artist**, un **Album** y un **Track** de prueba, y luego leerlos.
4. Mostrar un mensaje claro por consola, del tipo:

```
[OK] H2 + DDL inicializados y mapeos JPA verificados.
```

4) Guía específica – Validaciones / lógica sugerida

No es obligatorio implementar lógica de negocio compleja para este pre-enunciado, pero **se sugiere** incluir algunas validaciones simples para practicar, por ejemplo:

- En **Track**:
 - Método que devuelva la duración en minutos/segundos a partir de **MILLISECONDS**.
 - Método que valide si el precio unitario (**UNIT_PRICE**) es mayor que cero.

```
public double getDurationInMinutes() {  
    return milliseconds != null ? milliseconds / 60000.0 : 0.0;
```

```

    }

    public boolean hasValidPrice() {
        return unitPrice != null && unitPrice.doubleValue() > 0.0;
    }
}

```

- En **Invoice**:

- Método que verifique que **TOTAL** es consistente (en este pre-enunciado alcanza con un chequeo simple de que **TOTAL > 0**).

Estas validaciones pueden ser usadas en el **main** para armar un pequeño **self-check** de la estructura.

5) Estrategia sugerida (pensando en el día del parcial)

Se sugiere lo siguiente de acuerdo con lo que se solicitará el día del parcial:

- Proyecto Maven compilable con **mvn compile** y ejecutable con **mvn exec:java** como está documentado en los materiales y lo hemos trabajado en clase.
- El proyecto debería respetar lo expresado en la presente consigna:
 - Ejecutar a partir del **Main** que inicializa la BD y comprueba el mapeo.
 - Usar una **URL JDBC consistente** (**jdbc:h2:mem:database;DB_CLOSE_DELAY=-1**).
 - Mantener el **persistence.xml** en **META-INF** y el DDL en **resources/sql/database-ddl.sql**.
 - Utilizar los proveedores de **DataSource** y **EntityManager** definidos en **infra**.
- Si se opta por otra alternativa, algunos elementos podrían alterar el funcionamiento de los componentes documentados.

Recordatorio especial: El día del parcial se entregará un **bloque de datos** (por ejemplo, álbumes y sus datos, playlists y sus datos o invoices y sus datos) ya adaptado a la estructura que preparada para popular en la BD. El objetivo es llegar al parcial con:

- La **BD H2** completamente definida y funcionando.
- El **mapeo JPA** de todas las tablas listo.
- Los **proveedores de infraestructura** probados. De este modo, en el examen se puede hacer foco en **cargar/usuarios ese bloque de datos** y en las **consultas/procesos** solicitados.

Sobre el uso de git u otros repositorios (Disclaimer)

No se requiere el uso de un repositorio específico para la contención temporal de lo vertido en la construcción del proyecto a partir del presente pre-enunciado, sin embargo:

- Los docentes de la cátedra no se hacen responsables, ni tienen la obligación de brindar solución ante la pérdida del trabajo realizado.
- Los docentes de la cátedra no tienen responsabilidad de que el entorno de trabajo funcione ni que cargue el pre-enunciado que haya generado.
- La cátedra no se hace responsable el día de la evaluación sobre elementos del entorno de trabajo ni de las dependencias asociadas a este... esto es parte de la evaluación.

Recordatorio final: todo preparado en esta etapa busca que, el día del parcial, solo sea necesario construir la **carga/uso del bloque de datos** y las **consultas y procesos** que se pidan.

Éxitos a todos en el desarrollo del parcial 🎉 🎉 🎉