

Nombre del proyecto: Musical life

Manual de usuario

Integrantes:

* Jiménez Rodríguez Lizet
* [Alonso Pérez Antonio](http://antonioalonsoperez.com/)

Contenido

[Problemática y solución 3](#_Toc23939694)

[Desarrollo 4](#_Toc23939695)

[Diseño Base de datos 5](#_Toc23939696)

[Diccionario de datos: 7](#_Toc23939697)

[Patrones de diseño: 8](#_Toc23939698)

[Justificación de patrones de diseño: 9](#_Toc23939699)

[Diagramas de clase: 11](#_Toc23939700)

# Problemática y solución

La empresa “Gamdias” no cuenta con un sistema digital para la gestión de sus procesos, estos los realizan mediante pesados formularios en papel que una vez llenados pasan a la sala de autorización para poder ser enviados a un departamento que ingresan esta información a un sistema virtual.

Por lo cual se plantea este proyecto llamado “Musical Life” para la gestión de estos procesos a través de un sistema digital de fácil manipulación que permita almacenar de forma rápida los datos del artista, disquera y el control de gestión de álbumes. Otorgando la facilidad de realizar respaldos semanalmente para evitar perdida de información y que los datos se encuentren disponibles en el momento que los requiera, permitiendo generar reportes sobre los artistas y álbumes registrados con un filtro de **nacionalidad** de los artistas y **fecha de creación** de los álbumes registrados, en un formato universal para su visualización (PDF).

Cumpliendo con los estándares de calidad de software mencionados en la ISO 9126. Aplicando y respetando los criterios que esta maneja:

* Funcionalidad
* Fiabilidad
* Usabilidad
* Eficiencia
* Mantenibilidad
* Portabilidad

# Desarrollo

Definición de conceptos

Para comenzar con el desarrollo de este proyecto entramos un poco en el contexto sobre el enfoque del sistema. Este software funciona para una empresa que quiere gestionar la información de artistas, disqueras y el control de apertura de álbum. Para entender de forma clara se explican los siguientes conceptos.

Artista:

Un cantante es un artista que produce con su voz melodías, normalmente utilizando palabras que suelen rimar. También suele acompañarse de música instrumental. En sentido amplio, designa a quien usa su voz para realizar melodías musicales. También se utiliza para designar a aquellas personas que se dedican a hacer música a través del uso de la voz, pudiendo hacerlo sin la necesidad de instrumentos musicales, aunque en muchos casos estos sirven como acompañamientos.

**Disquera**:

Una compañía discográfica o casa discográfica, también conocida como sello discográfico, discográfica o disquera, es una empresa que se dedica a realizar grabaciones de música, así como su comercialización y distribución. Algunas de estas compañías tienen sus propios estudios de grabación y sus propios profesionales, para buscar el mejor sonido en la grabación de un álbum musical como productores discográficos.

Álbum:

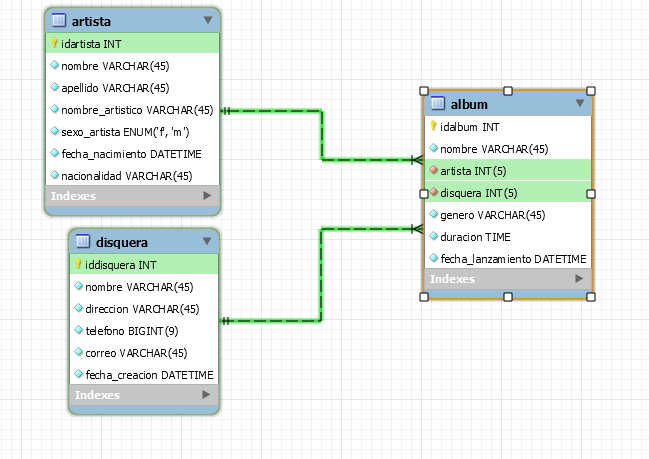
Un álbum musical consistente en una selección de canciones grabadas en un estudio de grabación. La diferencia de un álbum de estudio con un álbum recopilatorio (el cual también suele contener canciones de estudio), es que el álbum de estudio incluye temas nuevos grabados por el artista, mientras que el recopilatorio, precisamente compila material publicado con anterioridad y ya conocido. Usualmente no contienen grabaciones en vivo o remixes, y si los tiene suelen aparecer como bonus tracks, y componen una pequeña parte del álbum.

# Diseño Base de datos

Base de datos:

Una base de datos es un “almacén” que nos permite guardar grandes cantidades de información de forma organizada para que luego podamos encontrar y utilizar fácilmente.

Para el primer diseño de la base de datos se plantean 3 tablas, tabla.artista, tabla.disquera y tabla.album. Siendo esta ultima la que contiene la relación entre las 2 tablas anteriores para realizar el registro correspondiente. El diseño entidad relación quedo de la siguiente manera.

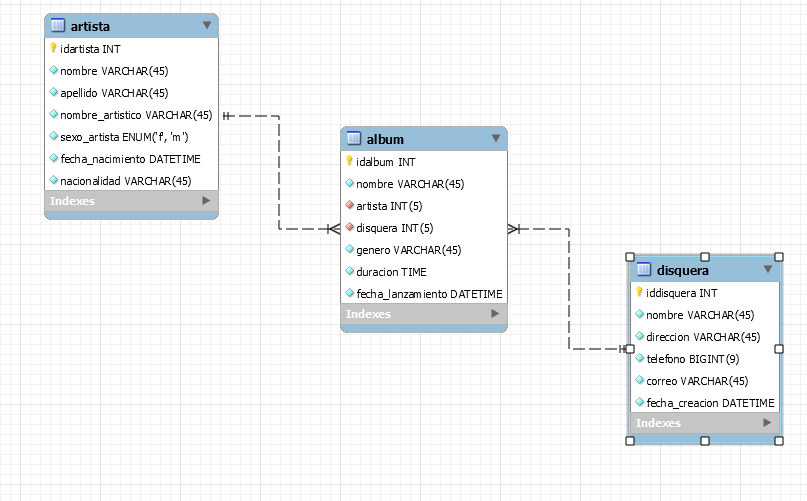


**Figura 1.** Diagrama entidad relación de la base de datos.

Normalización de una base de datos:

El proceso de normalización de una base de datos relacional consiste en aplicar una serie de reglas para evitar a futuro realizar consultas innecesariamente complejas. En otras palabras, están enfocadas en eliminar redundancias e inconsistencias de dependencia en el diseño de las tablas.

Una vez aplicadas las tres reglas normales para la normalización de la base de datos se realizan las modificaciones correspondientes y nuestro diagrama de base de datos normalizada queda de la siguiente manera:



**Figura 1.2.** Diagrama entidad relación de la base de datos normalizada.

Se adjunta a este documento el script de la base de datos (.sql) y el diccionario de datos de esta misma (.html).

# Diccionario de datos:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Artista | | | | | | | | |
|  | | | | | | | | |
| Column name | **DataType** | **PK** | **NN** | **UQ** | **BIN** | **UN** | **ZF** | **AI** |
| idartista | INT | ✔ | ✔ |  |  |  |  | ✔ |
| nombre | VARCHAR(45) |  | ✔ |  |  |  |  |  |
| apellido | VARCHAR(45) |  | ✔ |  |  |  |  |  |
| nombre\_artistico | VARCHAR(45) |  | ✔ |  |  |  |  |  |
| sexo\_artista | ENUM('f', 'm') |  | ✔ |  |  |  |  |  |
| fecha\_nacimiento | DATETIME |  | ✔ |  |  |  |  |  |
| nacionalidad | VARCHAR(45) |  | ✔ |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Album | | | | | | | | |
|  | | | | | | | | |
| Column name | **DataType** | **PK** | **NN** | **UQ** | **BIN** | **UN** | **ZF** | **AI** |
| idalbum | INT | ✔ | ✔ |  |  |  |  | ✔ |
| nombre | VARCHAR(45) |  | ✔ |  |  |  |  |  |
| artista | INT(5) |  | ✔ |  |  |  |  |  |
| disquera | INT(5) |  | ✔ |  |  |  |  |  |
| genero | VARCHAR(45) |  | ✔ |  |  |  |  |  |
| duracion | TIME |  | ✔ |  |  |  |  |  |
| fecha\_lanzamiento | DATETIME |  | ✔ |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Disquera | | | | | | | | | | |
|  | | | | | | | | | | |
| Column name | **DataType** | **PK** | **NN** | **UQ** | **BIN** | **UN** | **ZF** | **AI** | **Default** | **Comment** |
| iddisquera | INT | ✔ | ✔ |  |  |  |  | ✔ |  |  |
| nombre | VARCHAR(45) |  | ✔ |  |  |  |  |  |  |  |
| direccion | VARCHAR(45) |  | ✔ |  |  |  |  |  |  |  |
| telefono | BIGINT(9) |  | ✔ |  |  |  |  |  |  |  |
| correo | VARCHAR(45) |  | ✔ |  |  |  |  |  |  |  |
| fecha\_creacion | DATETIME |  | ✔ |  |  |  |  |  |  |  |

# Patrones de diseño:

Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Existen 3 categorias para los patrones de diseño que son las siguientes:

* Creación
* Comportamiento
* Estructura

Para este proyecto se implementan 3 patrones de diseño, uno por cada categoría y los seleccionados por parte de los desarrolladores son los siguientes:

* Singleton
* Facade
* Observer

# Justificación de patrones de diseño:

Singleton:

El patrón de diseño Singleton se implementa en la clase Conexión, esta clase es la encargada de interactuar con los objetos del servidor de base de datos PostgreSQL para poder aplicar las diferentes sentencias del sistema.

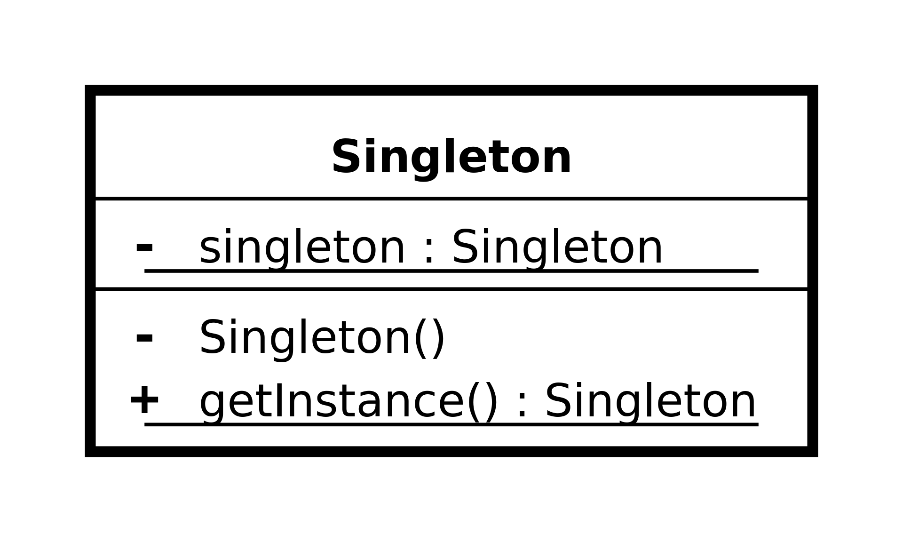
Este patrón su intención consiste en garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella.

El patrón singleton se implementa creando en nuestra clase un método que crea una instancia del objeto solo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con modificadores de acceso como protegido o privado).

Sus principales características son:

* La propia clase es responsable de crear la única instancia.
* Permite el acceso global a dicha instancia mediante un método de clase.
* Declara el constructor de clase como privado para que no sea instanciable directamente.
* Al estar internamente autoreferenciada, en lenguajes como Java, el recolector de basura no actúa.

Se implementa este patrón de diseño de tipo creación para crear un único objeto de la clase ConexionPostgresSingleton para poder crear un único objeto y llamarlo a cualquier clase que requiera tener acceso a este objeto para poder realizar las sentencias en la base de datos.



**Figura 1.3.** Estructura del patrón de diseño Singleton.

Facade:

De la categoría de patrones de Estructura se seleccionó Facade ya que es un tipo de patrón de diseño estructural. Viene motivado por la necesidad de estructurar un entorno de programación y reducir su complejidad con la división en subsistemas, minimizando las comunicaciones y dependencias entre estos.

Se aplicará el patrón fachada cuando se necesite proporcionar una interfaz simple para un subsistema complejo, o cuando se quiera estructurar varios subsistemas en capas, ya que las fachadas serían el punto de entrada a cada nivel. Otro escenario proclive para su aplicación surge de la necesidad de desacoplar un sistema de sus clientes y de otros subsistemas, haciéndolo más independiente, portable y reutilizable (esto es, reduciendo dependencias entre los subsistemas y los clientes).

El patrón Facade se implementa en este sistema por la utilidad que este representa, ya que parte de una clase principal y para su correcto funcionamiento depende de submodulos que están conectados entre si para poder complementar la clase principal del sistema.

Observer:

El objetivo principal de este patrón de diseño es definir una dependencia uno a muchos entre objetos, de tal forma que cuando el objeto cambie de estado, todos sus objetos dependientes sean notificados automáticamente. Se trata de desacoplar la clase de los objetos clientes del objeto, aumentando la modularidad del lenguaje, creando las mínimas dependencias y evitando bucles de actualización (espera activa o sondeo). En definitiva, normalmente, se usará el patrón observador cuando un elemento quiere estar pendiente de otro, sin tener que estar comprobando de forma continua si ha cambiado o no.

Se implementará este patrón de diseño en la clase Reportes, debido a que su estructura nos ayuda a saber el estado de un objeto. Este objeto se aplicará en la creación de reportes (documentos PDF con información extraída de la base de datos) utilizando hilos de ejecución asociamos este proceso a un objeto de tipo Observer para poder tener el valor del estado de creación del archivo en tiempo real para poder seguir con las diferentes acciones.

A su vez nos ayuda a validar

# Diagramas de clase:

Un diagrama de clases es una herramienta para comunicar el diseño de un programa que se creó para orientar objetos y que permite modelar relaciones entre diferentes entidades.

En UML, una clase se representa con un rectángulo que posee tres divisiones, nombre de la clase, abributos que tiene y mensajes que entiende.

En el primero de los cuadros se anota el nombre de la clase. Si es abstracta, se escribe en letra cursiva o también se utiliza un estereotipo como < > arriba del nombre de la clase.

En la segunda parte van los atributos o variables de instancia; las variables de clase van subrayados.

En el último cuadro se escriben las operaciones, es decir, los mensajes que puede entender. Lo importante es documentar los mensajes más relevantes y no todos los mensajes de un solo objeto.

Debemos tener en cuenta que una clase que no tiene comportamiento no está comunicando qué tipo de rol cumple en la solución, así que o está faltando definir qué es lo que le puedo pedir o entonces esa clase no debería estar en el diagrama.

En este proyecto se utilizan 11 clases para su completa funcionalidad de este mismo.

|  |  |
| --- | --- |
| Clase: | Función: |
| PanelAlbum | Muestra el diseño gráfico (formulario) de cada categoría. |
| PanelArtista |
| PanelDisquera |
| PanelReportes |
| PanelPrincipal | Ventana principal del sistema (main) |
| Album | Contienen todo el paradigma de la programación orientada a objetos y se realizan las acciones relacionadas con la base de datos. De cada categoría. |
| Artista |
| Disquera |
| Reportes | Clase donde se crean los reportes correspondientes a la acción del usuario. |
| ConexionDB | Clase intermediaria para conectar al servidor de base de datos PostgreSQL al sistema con la ayuda de un driver especifico. |
| InterfazBase | Clase abstracta pura creada para implementar los mismos métodos en diferentes clases según se requieran. |

**Tabla 1.** Descripción de clases del sistema.

