# Software para Gestión de eventos de Anime

Campers:

Robinson Mosquera Cubides

Hernán Méndez Guerrero

Campuslands

Grupo:

P1

Trainer:

Pedro Felipe Gómez Bonilla

30 de agosto del 2024

# Contenido

# Introducción

El proyecto "Software para la Gestión de Eventos de Anime" busca proporcionar una solución completa para la planificación y administración de eventos relacionados con la cultura del anime. Este sistema permitirá organizar y gestionar múltiples eventos de manera simultánea, abarcando aspectos esenciales como la gestión de boletos, actividades como concursos de cosplay y trivias, y la administración de comercios, incluyendo tiendas y restaurantes.

En cuanto a la creación y administración de eventos, la plataforma permitirá coordinar eventos paralelos, asignar roles al personal (seguridad, limpieza, encargados de tiendas), y gestionar el inventario necesario para su desarrollo, como decoraciones, sillas, luces y stands.

El módulo de boletos facilitará la administración de la venta de entradas para cada evento, permitiendo crear diferentes tipos de boletos, gestionar a los visitantes y generar informes financieros sobre las ventas.

Para las actividades del evento, como concursos y trivias, el software permitirá gestionar a los participantes, actividades y premios, así como generar reportes sobre el desempeño de cada actividad.

Finalmente, el sistema de comercios gestionará de manera eficiente las tiendas y restaurantes dentro del evento, abarcando la administración del personal, el seguimiento de pedidos, y la generación de reportes sobre el balance de ventas. Este software se está desarrollando con Java (JDK 17.0.7) para la lógica del sistema y MySQL 8.3.0 para la gestión de bases de datos.

# Caso de estudio

El Software para la Gestión de Eventos de Anime fue creado para enfrentar los desafíos logísticos y operativos en la organización de eventos temáticos. Esta solución facilita la gestión simultánea de varios eventos, abarcando desde la creación y administración de roles del personal hasta la gestión de inventarios, taquillas, y actividades como cosplay y trivia, así como comercios, incluyendo tiendas y restaurantes. Desarrollado en Java (JDK 17.0.7) y utilizando MySQL 8.3.0, el software centraliza la planificación, optimiza la coordinación del equipo, mejora la experiencia de los asistentes y proporciona informes financieros detallados. Los beneficios incluyen una mayor eficiencia operativa, reducción de costos y mejor transparencia contable, convirtiéndose en una herramienta eficaz para los organizadores de eventos de anime.

# Instalación

**1. Clonar el Repositorio desde GitHub**

- Abre la terminal de tu dispositivo:
    - **Windows**:
        - Presiona las teclas **Windows + R**.
        - Escribe cmd y presiona **Enter**.
    - **Ubuntu**:
        - Presiona **Ctrl + Alt + T**.
- Clona el repositorio utilizando el comando:

```
git clone <URL del repositorio>
```

**2. Instalar MySQL**

- Si no tienes MySQL instalado, descárgalo e instálalo desde la página oficial de MySQL.
- Durante la instalación, configura un **usuario** y una **contraseña segura** que recuerdes para más adelante.

**3. Crear la Base de Datos e Importar las Tablas**

- Abre **MySQL Workbench** u otra herramienta de gestión de bases de datos de tu preferencia.
- Crea una nueva base de datos ejecutando la siguiente instrucción:

```
CREATE DATABASE bicgnrjz9nek32oc7te0;
```

- Importa los archivos **.sql** que clonaste del repositorio a tu herramienta de gestión de bases de datos para crear las tablas necesarias.

**4. Instalar JDK (Java Development Kit)**

- Si no tienes el JDK instalado, descárgalo e instálalo desde la página oficial de Oracle en su versión **17.0.7**.

**5. Instalar NetBeans**

- Descarga e instala **NetBeans** desde la página oficial de NetBeans.

**6. Configurar la Conexión de la Base de Datos en el Proyecto**

- Abre el proyecto en NetBeans.
- Dirígete al archivo de configuración de la conexión a la base de datos, normalmente llamado Config.properties.

- Asegúrate de que los detalles de la conexión (URL, usuario, contraseña) coincidan con la configuración de MySQL que configuraste durante la instalación.

**7. Agregar la Biblioteca MySQL Connector a NetBeans**

- Descarga el **MySQL Connector JDBC** desde la página oficial.
- En NetBeans, ve a **Tools > Libraries**.
- Haz clic en **New Library**, dale un nombre (por ejemplo, "MySQL Connector").
- Haz clic en **Add JAR/Folder** y selecciona el archivo JAR del MySQL Connector que descargaste.
- Haz clic en **OK**.

**8. Agregar la Biblioteca al Proyecto**

- En NetBeans, haz clic derecho sobre el proyecto y selecciona **Properties**.
- Ve a la pestaña **Libraries > Compile > Add Library**.
- Selecciona la biblioteca "MySQL Connector" que creaste y haz clic en **Add Library**.

**9. Compilar y Ejecutar el Proyecto**

- En NetBeans, haz clic derecho sobre el proyecto y selecciona **Clean and Build** para compilarlo.
- Una vez completada la compilación, haz clic derecho sobre el proyecto nuevamente y selecciona **Run** para ejecutar el proyecto.

# Planificación

**Objetivo del proyecto:**

Desarrollar un software integral para la gestión de eventos de anime, enfocado en la creación y administración eficiente de eventos, taquillas, actividades y comercios. Este sistema optimizará la organización mediante la automatización de procesos, la centralización de la información y la generación de reportes detallados. El objetivo del proyecto es mejorar la experiencia tanto de los organizadores como de los asistentes, asegurando un control eficiente de los recursos y una gestión financiera clara y transparente, facilitando así una toma de decisiones más informada.

**Estructura de la base de datos:**

**1. Tabla Locations**

- **Propósito**: Almacenar la ubicación física de los eventos.
- **Campos**:
  - id: Identificador único de la ubicación.
  - country, city, address: Información detallada de la ubicación.
  - **Restricción**: UNIQUE (country, city, address) asegura que no se repitan ubicaciones idénticas.

**2. Tabla AgeClassifications**

- **Propósito**: Almacenar las clasificaciones de edad aplicables a eventos y boletos.
- **Campos**:
  - id: Identificador único.
  - classification: Clasificación de edad (ej. "Todas las edades", "18+").
  - **Restricción**: UNIQUE (classification) evita duplicación.

**3. Tabla EmployeeStatuses**

- **Propósito**: Almacenar los posibles estados laborales de los empleados (ej. asignado, despedido).
- **Campos**:
    - id: Identificador único.
    - status_name: Nombre del estado.
    - **Restricción**: UNIQUE (status_name) asegura que no haya estados duplicados.

**4. Tabla InventoryItemStatuses**

- **Propósito**: Almacenar los diferentes estados de los ítems en inventario (ej. en uso, en almacén).
- **Campos**:
    - id: Identificador único.
    - status_name: Nombre del estado.
    - **Restricción**: UNIQUE (status_name) garantiza que los nombres no se repitan.

**5. Tabla GeneralCategories**

- **Propósito**: Almacenar categorías generales para eventos, platos, tiendas y actividades.
- **Campos**:
    - id: Identificador único.
    - category_name: Nombre de la categoría.
    - type: Tipo de categoría (EVENT, DISH, SHOP, ACTIVITY).
    - **Restricción**: UNIQUE (category_name, type) evita duplicados dentro de cada tipo de categoría.

**6. Tabla Roles**

- **Propósito**: Almacenar los roles asignables a los empleados (ej. seguridad, encargado de tienda).

- **Campos**:
    - id: Identificador único.
    - name: Nombre del rol.
    - **Restricción**: UNIQUE (name) para evitar duplicación de roles.

**7. Tabla Activities**

- **Propósito**: Almacenar las actividades generales disponibles en los eventos (ej. cosplay, trivia).
- **Campos**:
    - id: Identificador único.
    - name: Nombre de la actividad.
    - **Restricción**: UNIQUE (name) previene duplicación.

**8. Tabla DishTypes**

- **Propósito**: Almacenar los diferentes tipos de platos ofrecidos en los restaurantes (ej. entrada, plato fuerte).
- **Campos**:
    - id: Identificador único.
    - type_name: Nombre del tipo de plato.
    - **Restricción**: UNIQUE (type_name) asegura que no haya tipos de platos repetidos.

**9. Tabla Organizers**

- **Propósito**: Almacenar la información de los organizadores de eventos.
- **Campos**:
    - id: Identificador único.
    - name: Nombre del organizador.
    - contact_info: Información de contacto.
    - **Restricción**: UNIQUE (name) para evitar la duplicación de organizadores.

**10. Tabla Events**

- **Propósito**: Almacenar la información de los eventos.
- **Campos**:
    - id: Identificador único.
    - name: Nombre del evento.
    - date_time: Fecha y hora del evento.
    - organizer_id: Referencia al organizador (tabla Organizers).
    - age_classification_id: Referencia a la clasificación de edad (tabla AgeClassifications).
    - status: Estado del evento (ej. activo, finalizado).
    - **Restricciones**:
        - UC_NameDate: Evita que dos eventos con el mismo nombre coincidan en la misma fecha.
        - FOREIGN KEY (organizer_id): Referencia a Organizers.
        - FOREIGN KEY (age_classification_id): Referencia a AgeClassifications.

**11. Tabla EventLocations**

- **Propósito**: Relacionar eventos con sus ubicaciones.
- **Campos**:
    - event_id: Referencia al evento (Events).
    - location_id: Referencia a la ubicación (Locations).
    - **Restricción**: PRIMARY KEY (event_id, location_id) para evitar duplicados.

**12. Tabla EventCapacities**

- **Propósito**: Almacenar la capacidad máxima de personas, tiendas y restaurantes para un evento.
- **Campos**:
    - id: Identificador único.
    - event_id: Referencia al evento (Events).

- max_capacity_people, max_capacity_shops, max_capacity_restaurants: Capacidades máximas.
- **Restricción**: FOREIGN KEY (event_id) referencia a la tabla Events.

**13. Tabla RoleActivityAssignments**

- **Propósito**: Asignar actividades específicas a los roles.
- **Campos**:
  - role_id: Referencia al rol (Roles).
  - activity_id: Referencia a la actividad (Activities).
  - assigned_date: Fecha de asignación.
  - **Restricción**: PRIMARY KEY (role_id, activity_id) para evitar duplicados.

**14. Tabla Employees**

- **Propósito**: Almacenar la información de los empleados.
- **Campos**:
  - id: Identificador único.
  - name: Nombre del empleado.
  - identification: Documento de identificación único.
  - birth_date: Fecha de nacimiento.
  - role_id: Referencia al rol (Roles).
  - status_id: Referencia al estado (EmployeeStatuses).
  - **Restricciones**:
    - FOREIGN KEY (role_id): Referencia a la tabla Roles.
    - FOREIGN KEY (status_id): Referencia a la tabla EmployeeStatuses.

**15. Tabla EmployeeEventAssignments**

- **Propósito**: Asignar empleados a eventos específicos.
- **Campos**:
  - id: Identificador único.

    - employee_id: Referencia al empleado (Employees).
    - event_id: Referencia al evento (Events).
    - **Restricciones**:
        - FOREIGN KEY (employee_id): Referencia a la tabla Employees.
        - FOREIGN KEY (event_id): Referencia a la tabla Events.

**16. Tabla EmployeeStatusHistory**

- **Propósito**: Mantener un historial de cambios de estado de los empleados.
- **Campos**:
    - id: Identificador único.
    - employee_id: Referencia al empleado (Employees).
    - status_id: Referencia al estado (EmployeeStatuses).
    - change_date: Fecha del cambio de estado.
    - **Restricciones**:
        - FOREIGN KEY (employee_id): Referencia a la tabla Employees.
        - FOREIGN KEY (status_id): Referencia a la tabla EmployeeStatuses.

**17. Tabla InventoryItems**

- **Propósito**: Gestionar el inventario de elementos de utilería del evento.
- **Campos**:
    - id: Identificador único.
    - name: Nombre del ítem.
    - quantity: Cantidad disponible.
    - status_id: Referencia al estado del ítem (InventoryItemStatuses).
    - event_id: Referencia al evento (Events).
    - **Restricciones**:
        - FOREIGN KEY (status_id): Referencia a la tabla InventoryItemStatuses.
        - FOREIGN KEY (event_id): Referencia a la tabla Events.

**18. Tabla TicketBooths**

- **Propósito**: Gestionar las taquillas de los eventos.
- **Campos**:
    - id: Identificador único.
    - event_id: Referencia al evento (Events).
    - location: Ubicación de la taquilla.
    - contact_number: Número de contacto.
    - manager_id: Referencia al empleado encargado (Employees).
    - **Restricciones**:
        - FOREIGN KEY (event_id): Referencia a la tabla Events.
        - FOREIGN KEY (manager_id): Referencia a la tabla Employees.

**19. Tabla Visitors**

- **Propósito**: Gestionar la información de los visitantes del evento.
- **Campos**:
    - id: Identificador único.
    - name: Nombre del visitante.
    - identification_document: Documento de identificación único.
    - gender: Género del visitante.
    - birth_date: Fecha de nacimiento.
    - email: Correo electrónico.
    - phone_number: Número de teléfono.

**20. Tabla TicketStatuses**

- **Propósito**: Almacenar los estados posibles de un ticket (pagado, reservado).
- **Campos**:
    - id: Identificador único.
    - status: Estado del ticket.

**21. Tabla Tickets**

- **Propósito**: Gestionar los boletos emitidos para el evento.
- **Campos:**
    - id: Identificador único.
    - name: Nombre del boleto.
    - price: Precio del boleto.
    - age_classification_id: Referencia a la clasificación de edad (AgeClassifications).
    - additional_cost: Costos adicionales para participar en actividades.
    - status_id: Referencia al estado del boleto (TicketStatuses).
    - ticket_booth_id: Referencia a la taquilla (TicketBooths).
    - customer_id: Referencia al cliente (Visitors).
- **Restricciones**:
    - FOREIGN KEY (age_classification_id): Referencia a la tabla AgeClassifications.
    - FOREIGN KEY (status_id): Referencia a la tabla TicketStatuses.
    - FOREIGN KEY (ticket_booth_id): Referencia a la tabla TicketBooths.
    - FOREIGN KEY (customer_id): Referencia a la tabla Visitors.

**22. Tabla EventActivities**

- **Propósito**: Gestionar las actividades realizadas en los eventos.
- **Campos**:
    - id: Identificador único.
    - name: Nombre de la actividad.
    - type: Tipo de actividad (Cosplay, Trivia).

    - number_of_participants: Número de participantes permitidos.
    - event_id: Referencia al evento (Events).
    - start_time: Hora de inicio.
    - manager_id: Referencia al empleado encargado (Employees).
    - category_id: Referencia a la categoría (GeneralCategories).
- **Restricciones**:
    - FOREIGN KEY (event_id): Referencia a la tabla Events.
    - FOREIGN KEY (manager_id): Referencia a la tabla Employees.
    - FOREIGN KEY (category_id): Referencia a la tabla GeneralCategories.

**23. Tabla ActivityParticipants**

- **Propósito**: Gestionar la participación de visitantes en actividades.
- **Campos**:
    - id: Identificador único.
    - activity_id: Referencia a la actividad (EventActivities).
    - visitor_id: Referencia al visitante (Visitors).
    - is_winner: Indicador de si el participante es ganador.
- **Restricciones**:
    - FOREIGN KEY (activity_id): Referencia a la tabla EventActivities.
    - FOREIGN KEY (visitor_id): Referencia a la tabla Visitors.

**24. Tabla Prizes**

- **Propósito**: Gestionar los premios asignados a las actividades.
- **Campos**:

- id: Identificador único.
- type: Tipo de premio.
- description: Descripción del premio.
- value: Valor del premio.
- status: Estado del premio (disponible, entregado).
- activity_id: Referencia a la actividad (EventActivities).
- participant_id: Referencia al participante (ActivityParticipants).

- **Restricciones**:
  - FOREIGN KEY (activity_id): Referencia a la tabla EventActivities.
  - FOREIGN KEY (participant_id): Referencia a la tabla ActivityParticipants.

**25. Tabla Shops**

- **Propósito**: Gestionar las tiendas y restaurantes en los eventos.
- **Campos**:
  - id: Identificador único.
  - name: Nombre de la tienda o restaurante.
  - category: Categoría (SHOP, RESTAURANT).
  - event_id: Referencia al evento (Events).
  - manager_id: Referencia al empleado encargado (Employees).
- **Restricciones**:
  - FOREIGN KEY (event_id): Referencia a la tabla Events.
  - FOREIGN KEY (manager_id): Referencia a la tabla Employees.

**26. Tabla Products**

- **Propósito**: Gestionar los productos de las tiendas.
- **Campos**:
    - id: Identificador único.
    - shop_id: Referencia a la tienda (Shops).
    - name: Nombre del producto.
    - description: Descripción del producto.
    - manufacturer: Fabricante.
    - type: Tipo de producto.
    - available_quantity: Cantidad disponible.
    - price: Precio.
- **Restricciones**:
    - FOREIGN KEY (shop_id): Referencia a la tabla Shops.

**27. Tabla Dishes**

- **Propósito**: Gestionar los platos en los restaurantes.
- **Campos**:
    - id: Identificador único.
    - restaurant_id: Referencia al restaurante (Shops).
    - name: Nombre del plato.
    - price: Precio del plato.
    - type_id: Referencia al tipo de plato (DishTypes).
    - preparation_time: Tiempo de preparación.
- **Restricciones**:
    - FOREIGN KEY (restaurant_id): Referencia a la tabla Shops.

- FOREIGN KEY (type_id): Referencia a la tabla DishTypes.

**28. Tabla Ingredients**

- **Propósito**: Gestionar los ingredientes utilizados en los platos.
- **Campos**:
  - id: Identificador único.
  - name: Nombre del ingrediente.
  - available_quantity: Cantidad disponible.

**29. Tabla DishIngredients**

- **Propósito**: Relacionar platos con los ingredientes necesarios.
- **Campos**:
  - dish_id: Referencia al plato (Dishes).
  - ingredient_id: Referencia al ingrediente (Ingredients).
  - quantity_required: Cantidad requerida del ingrediente.
- **Restricciones**:
  - FOREIGN KEY (dish_id): Referencia a la tabla Dishes.
  - FOREIGN KEY (ingredient_id): Referencia a la tabla Ingredients.
  - PRIMARY KEY (dish_id, ingredient_id): Para evitar duplicados.

**30. Tabla CashRegisters**

- **Propósito**: Gestionar las cajas registradoras en los comercios.
- **Campos**:
  - id: Identificador único.

    - status: Estado de la caja (activo, inactivo).
    - operator_id: Referencia al operario (Employees).
    - opening_amount: Monto de apertura.
    - closing_amount: Monto de cierre.
- **Restricciones**:
    - FOREIGN KEY (operator_id): Referencia a la tabla Employees.

**31. Orders**

- **Propósito**: Administrar los pedidos hechos en las tiendas y restaurantes.
- **Campos**:
    - id: Identificador único.
    - visitor_id: Referencia al visitante (Visitors).
    - shop_id: Referencia a la tienda (Shops).
    - cash_register_id: Referencia a la caja registradora (CashRegisters).
    - cashier_id: Referencia al cajero (Employees).
    - total_value: Valor total del pedido.
    - status: Estado del pedido (registrado, pagado, entregado).
- **Restricciones**:
    - FOREIGN KEY (visitor_id): Referencia a la tabla Visitors.
    - FOREIGN KEY (shop_id): Referencia a la tabla Shops.
    - FOREIGN KEY (cash_register_id): Referencia a la tabla CashRegisters.
    - FOREIGN KEY (cashier_id): Referencia a la tabla Employees.

**32. Judges**

- **Propósito**: Administrar a los jueces de los eventos.
- **Campos**:
  - id: Identificador único.
  - name: Nombre del juez.
  - event_id: Referencia al evento (Events).
- **Restricciones**:
  - FOREIGN KEY (event_id): Referencia a la tabla Events.

**33. CosplayScores**

- **Propósito**: Registrar los puntajes asignados a los participantes de cosplay.
- **Campos**:
  - id: Identificador único.
  - participant_id: Referencia al participante (ActivityParticipants).
  - judge_id: Referencia al juez (Judges).
  - score: Puntaje dado por el juez.
- **Restricciones**:
  - FOREIGN KEY (participant_id): Referencia a la tabla ActivityParticipants.
  - FOREIGN KEY (judge_id): Referencia a la tabla Judges.

**34. TriviaQuestions**

- **Propósito**: Administrar las preguntas para los concursos de trivia.
- **Campos**:
  - id: Identificador único.
  - question: Texto de la pregunta.

    - correct_answer: Respuesta correcta.
    - category: Categoría de la pregunta.
    - difficulty: Nivel de dificultad de la pregunta (Fácil, Intermedio, Difícil).
    - event_id: Referencia al evento (Events).
    - category_id: Referencia a la categoría (GeneralCategories).
- **Restricciones**:
    - FOREIGN KEY (event_id): Referencia a la tabla Events.
    - FOREIGN KEY (category_id): Referencia a la tabla GeneralCategories.

**35. Discounts**

- **Propósito**: Administrar descuentos y promociones en las tiendas.
- **Campos**:
    - id: Identificador único.
    - shop_id: Referencia a la tienda (Shops).
    - product_id: Referencia al producto (Products).
    - discount_type: Tipo de descuento (PERCENTAGE, FIXED_AMOUNT, BUNDLE).
    - discount_value: Valor del descuento.
    - start_date: Fecha de inicio.
    - end_date: Fecha de finalización.
- **Restricciones**:
    - FOREIGN KEY (shop_id): Referencia a la tabla Shops.
    - FOREIGN KEY (product_id): Referencia a la tabla Products.

**36. TriviaRounds**

- **Propósito**: Administrar las rondas de los concursos de trivia.
- **Campos**:
    - id: Identificador único.
    - event_id: Referencia al evento (Events).
    - round_number: Número de la ronda.
    - participant1_id: Referencia al primer participante (ActivityParticipants).
    - participant2_id: Referencia al segundo participante (ActivityParticipants).
    - winner_id: Referencia al ganador de la ronda (ActivityParticipants).
- **Restricciones**:
    - FOREIGN KEY (event_id): Referencia a la tabla Events.
    - FOREIGN KEY (participant1_id): Referencia a la tabla ActivityParticipants.
    - FOREIGN KEY (participant2_id): Referencia a la tabla ActivityParticipants.
    - FOREIGN KEY (winner_id): Referencia a la tabla ActivityParticipants.

**37. CosplayCategories**

- **Propósito**: Administrar las categorías de los concursos de cosplay.
- **Campos**:
    - id: Identificador único.
    - name: Nombre de la categoría.
    - description: Descripción de la categoría.

**38. OrderDetails**

- **Propósito**: Administrar los detalles de los pedidos realizados en tiendas o restaurantes.

- **Campos**:
    - id: Identificador único.
    - order_id: Referencia al pedido (Orders).
    - product_id: Referencia al producto (Products).
    - quantity: Cantidad del producto.
    - unit_price: Precio unitario del producto.
- **Restricciones**:
    - FOREIGN KEY (order_id): Referencia a la tabla Orders.
    - FOREIGN KEY (product_id): Referencia a la tabla Products.

# Justificación de las tablas

**1. Events**

La tabla Events es fundamental para almacenar toda la información relativa a los eventos, incluyendo su nombre, descripción, fechas y estado. Esta tabla actúa como el núcleo del sistema de gestión de eventos, permitiendo una organización clara y centralizada de cada evento que se realiza. Los campos de esta tabla permiten gestionar todos los aspectos fundamentales de un evento, como la ubicación, fechas importantes y la disponibilidad de recursos.

**2. Locations**

La tabla Locations gestiona los datos de las ubicaciones donde se llevan a cabo los eventos. Incluye detalles como la dirección y el tipo de ubicación, lo que facilita la planificación y logística al asegurar que cada evento se asocie con una ubicación específica y adecuada para su tamaño y tipo de actividades.

**3. Shops**

La tabla Shops registra la información de las tiendas participantes en los eventos, como su nombre, descripción y ubicación dentro del evento. Esto permite gestionar y coordinar las tiendas que ofrecen productos o servicios durante el evento, mejorando la experiencia del visitante al proporcionar un acceso organizado a los diferentes puntos de venta.

**4. Products**

La tabla Products almacena los detalles de los productos disponibles en las tiendas y restaurantes, como el nombre, descripción y precio. Esta tabla es crucial para la gestión de inventario y ventas, permitiendo un seguimiento preciso de los productos ofrecidos y sus características.

**5. CashRegisters**

La tabla CashRegisters mantiene la información sobre las cajas registradoras utilizadas en las tiendas y restaurantes. Incluye el número de caja y su ubicación, lo cual es importante para la gestión de las transacciones financieras y el seguimiento de las operaciones en cada punto de venta.

**6. GeneralCategories**

La tabla GeneralCategories clasifica las categorías generales de preguntas para concursos de trivia. Facilita la organización y filtrado de preguntas en función de su categoría, permitiendo una mejor estructuración y personalización de los concursos.

**7. Roles**

La tabla Roles define los diferentes roles disponibles dentro del sistema, como administrador o cajero. Este registro es fundamental para la asignación de tareas y la gestión de permisos, garantizando que cada usuario tenga el acceso adecuado a las funcionalidades del sistema según su rol.

**8. Employees**

La tabla Employees contiene la información de los empleados que trabajan en los eventos, incluyendo datos personales y de contacto. Esta tabla es crucial para la administración del

personal, facilitando la asignación de tareas y la gestión de recursos humanos durante el evento.

**9. ActivityParticipants**

La tabla ActivityParticipants gestiona la información de los participantes en diversas actividades del evento, como concursos y actividades recreativas. Permite un seguimiento detallado de los participantes y su participación en cada actividad, lo que es esencial para la organización y evaluación de eventos.

**10. EmployeeStatuses**

La tabla EmployeeStatuses registra los diferentes estados posibles de los empleados, como activo o inactivo. Esto es importante para la gestión del estado del personal y para mantener un registro actualizado de su disponibilidad y estado de empleo.

**11. EventLocations**

La tabla EventLocations relaciona eventos con sus ubicaciones específicas. Esto permite vincular de manera efectiva cada evento con una ubicación concreta, facilitando la planificación y logística al asegurar que cada evento esté asociado correctamente con su lugar de realización.

**12. EventCapacities**

La tabla EventCapacities almacena la capacidad máxima para personas, tiendas y restaurantes en un evento. Este registro es fundamental para gestionar el tamaño y la capacidad del evento, garantizando que el espacio sea adecuado para el número de asistentes y el tipo de actividades.

**13. RoleActivityAssignments**

La tabla RoleActivityAssignments asigna actividades específicas a los diferentes roles dentro del sistema. Permite una gestión eficiente de las tareas y responsabilidades al asegurar que cada rol tenga asignadas las actividades correspondientes, facilitando la organización y ejecución de estas.

**14. Employees**

La tabla Employees almacena la información detallada de los empleados, incluyendo datos personales, identificación y estado. Esta tabla es esencial para la administración del personal, facilitando la gestión de recursos humanos y la asignación de tareas y responsabilidades.

**15. EmployeeEventAssignments**

La tabla EmployeeEventAssignments asigna empleados a eventos específicos. Esto facilita la planificación y coordinación del personal, asegurando que los empleados estén correctamente asignados a los eventos en los que participan y puedan cumplir con sus tareas de manera efectiva.

**16. EmployeeStatusHistory**

La tabla EmployeeStatusHistory mantiene un registro histórico de los cambios en el estado de los empleados. Esto es importante para rastrear la evolución del estado de empleo de cada empleado a lo largo del tiempo, proporcionando un historial detallado para referencia futura.

**17. InventoryItems**

La tabla InventoryItems gestiona el inventario de elementos de utilería para eventos. Incluye detalles como el nombre, cantidad y estado del ítem, permitiendo una administración eficiente del inventario y asegurando que los recursos estén disponibles y en buen estado para el evento.

**18. TicketBooths**

La tabla TicketBooths registra la información sobre las taquillas en los eventos, incluyendo su ubicación, número de contacto y el empleado encargado. Esto es crucial para la gestión de la venta de entradas y la atención al público durante el evento.

**19. Visitors**

La tabla Visitors almacena la información de los visitantes del evento, como nombre, documento de identificación y datos de contacto. Esta tabla es fundamental para gestionar el registro y seguimiento de los asistentes, mejorando la organización y la experiencia del evento.

**20. TicketStatuses**

La tabla TicketStatuses define los posibles estados de un ticket, como pagado o reservado. Permite gestionar el estado de los tickets y asegura un seguimiento adecuado del proceso de venta y validación de entradas.

**21. Orders**

La tabla Orders gestiona los pedidos realizados en las tiendas y restaurantes durante los eventos. Incluye información como el visitante que realizó el pedido, la tienda, la caja registradora, el cajero y el valor total del pedido, así como el estado del pedido. Esto permite una administración precisa y un seguimiento adecuado de cada pedido.

**22. Judges**

La tabla Judges gestiona la información de los jueces en los eventos. Incluye datos como el nombre del juez y el evento al que está asignado. Esto facilita la organización de los concursos y actividades que requieren evaluación por parte de jueces.

**23. CosplayScores**

La tabla CosplayScores almacena los puntajes asignados a los participantes en concursos de cosplay. Incluye información sobre el participante, el juez que otorgó el puntaje y el valor del puntaje. Esto permite el seguimiento y cálculo de los resultados en competiciones de cosplay.

**24. TriviaQuestions**

La tabla TriviaQuestions gestiona las preguntas para concursos de trivia. Incluye el texto de la pregunta, la respuesta correcta, la categoría, la dificultad y el evento al que pertenece. Esto permite organizar y personalizar los concursos de trivia en función de las preguntas disponibles.

**25. Discounts**

La tabla Discounts almacena información sobre descuentos y promociones aplicables en tiendas durante los eventos. Incluye detalles como el tipo y valor del descuento, así como las fechas de inicio y finalización. Esto facilita la gestión de ofertas especiales y promociones en los puntos de venta.

**26. TriviaRounds**

La tabla TriviaRounds gestiona las rondas de los concursos de trivia. Incluye información sobre el evento, el número de ronda, los participantes y el ganador. Esto permite la organización y seguimiento de las diferentes rondas en los concursos de trivia.

**27. CosplayCategories**

La tabla CosplayCategories gestiona las categorías en los concursos de cosplay. Incluye el nombre y la descripción de cada categoría, lo que facilita la organización y clasificación de los participantes en función de las categorías disponibles.

**28. OrderDetails**

La tabla OrderDetails gestiona los detalles de los pedidos realizados en tiendas o restaurantes, incluyendo el producto, cantidad y precio unitario. Esto permite desglosar los ítems de cada pedido y calcular el total de manera precisa.

**29. ActivityParticipants**

La tabla ActivityParticipants gestiona la información de los participantes en diferentes actividades dentro de un evento. Incluye datos como el nombre y la identificación del participante. Esto permite un seguimiento organizado y eficiente de los participantes en cada actividad, facilitando la administración de las competencias y eventos recreativos.

**30. EventCapacities**

La tabla EventCapacities almacena la capacidad máxima para diferentes aspectos del evento, como el número de personas, tiendas y restaurantes. Esto es crucial para asegurar que el evento pueda manejar adecuadamente el número de asistentes y los recursos disponibles, garantizando una experiencia fluida y sin sobrecarga.

**31. RoleActivityAssignments**

La tabla RoleActivityAssignments asigna actividades específicas a los roles definidos en el sistema. Incluye información sobre el rol, la actividad y la fecha de asignación. Esto asegura que cada rol tenga tareas claramente definidas y permite una gestión eficiente de las actividades relacionadas con cada rol.

**32. EmployeeEventAssignments**

La tabla EmployeeEventAssignments asigna empleados a eventos específicos. Registra el empleado y el evento al que está asignado. Esto permite la planificación y coordinación del personal para cada evento, asegurando que el equipo adecuado esté disponible y asignado según las necesidades del evento.

**33. EmployeeStatusHistory**

La tabla EmployeeStatusHistory mantiene un registro histórico de los cambios en el estado de los empleados. Incluye datos sobre el empleado, el estado y la fecha del cambio. Esto proporciona un historial detallado que ayuda a rastrear la evolución del estado de empleo a lo largo del tiempo.

**34. InventoryItems**

La tabla InventoryItems gestiona los elementos de utilería para los eventos. Incluye información como el nombre del ítem, la cantidad disponible y su estado. Esto permite un control preciso del inventario y asegura que los recursos necesarios para el evento estén disponibles y en condiciones adecuadas.

**35. TicketBooths**

La tabla TicketBooths registra la información de las taquillas en los eventos, incluyendo su ubicación, número de contacto y el empleado encargado. Esto facilita la administración de la venta de entradas y la atención al público, asegurando que las taquillas estén bien gestionadas y operativas durante el evento.

**36. Visitors**

La tabla Visitors almacena los detalles de los visitantes del evento, como nombre, documento de identificación y datos de contacto. Esto es esencial para gestionar el registro y el seguimiento de los asistentes, permitiendo una mejor organización y una experiencia más personalizada durante el evento.

**37. TicketStatuses**

La tabla TicketStatuses define los posibles estados de un ticket, como pagado o reservado. Esto permite gestionar el estado de los tickets de manera eficiente, asegurando un seguimiento adecuado del proceso de venta y la validación de las entradas.

**38. OrderDetails**

La tabla OrderDetails gestiona los detalles de los pedidos realizados en tiendas o restaurantes. Incluye información sobre el pedido, el producto, la cantidad y el precio unitario. Esto permite desglosar los ítems de cada pedido, calcular el total y garantizar una gestión precisa de las ventas y el inventario.

# Ejecución

## MySQL

### Construcción del modelo conceptual

El modelo conceptual de la base de datos del "software para gestión de eventos de anime" está diseñado para gestionar de manera eficiente todos los aspectos relacionados con la organización de eventos de anime, incluyendo actividades como cosplay y trivia, tiendas, pedidos, taquillas y la administración del personal involucrado. A continuación, describo el propósito y las relaciones de cada tabla en el sistema:

- Events
    - *Descripción*: Almacena información sobre los eventos de anime organizados, gestionando todos los detalles relevantes.
    - *Atributos*: ID, Nombre del evento, Fecha de inicio, Fecha de fin, Descripción, Estado.

- Locations
    - *Descripción*: Registra las ubicaciones donde se realizan los eventos o actividades relacionadas.
    - *Atributos*: ID, Nombre de la ubicación, Dirección, Capacidad.

- Activities

  - *Descripción*: Gestiona las actividades específicas que se desarrollan durante los eventos, como concursos o trivias.
  - *Atributos*: ID, Nombre de la actividad, Descripción, Duración.

- Roles

  - *Descripción*: Define los roles de las personas involucradas en la organización de los eventos, como jueces, empleados o administradores.
  - *Atributos*: ID, Nombre del rol, Descripción.

- Shops

  - *Descripción*: Almacena información sobre las tiendas o comercios que participan en los eventos, vendiendo productos o servicios a los asistentes.
  - *Atributos*: ID, Nombre de la tienda, Tipo de tienda, Ubicación.

- Restaurants

  - *Descripción*: Registra los restaurantes que ofrecen servicios durante los eventos, incluyendo detalles del menú y ubicación.
  - *Atributos*: ID, Nombre del restaurante, Tipo de cocina, Ubicación.

- EventLocations

  - *Descripción*: Relaciona los eventos con las ubicaciones específicas donde se desarrollan.
  - *Atributos*: event_id, location_id.

- EventCapacities

  - *Descripción*: Gestiona las capacidades máximas de personas, tiendas y restaurantes permitidos en cada evento.
  - *Atributos*: ID, event_id, Capacidad máxima de personas, Capacidad máxima de tiendas, Capacidad máxima de restaurantes.

- RoleActivityAssignments

  - *Descripción*: Asigna actividades específicas a los diferentes roles dentro de los eventos, asegurando una correcta distribución de responsabilidades.
  - *Atributos*: role_id, activity_id, Fecha de asignación.

- Employees
    - *Descripción*: Almacena la información de los empleados que participan en los eventos, incluyendo sus roles y estado dentro de la organización.
    - *Atributos*: ID, Nombre, Documento de identificación, Fecha de nacimiento, role_id.

- EmployeeEventAssignments
    - *Descripción*: Relaciona a los empleados con los eventos en los que están asignados para trabajar.
    - *Atributos*: ID, employee_id, event_id.

- EmployeeStatusHistory
    - *Descripción*: Mantiene un registro histórico de los cambios en los estados de los empleados dentro de la organización.
    - *Atributos*: ID, employee_id, status_id, Fecha de cambio.

- InventoryItems
    - *Descripción*: Gestiona los elementos de utilería e inventario utilizados durante los eventos.
    - *Atributos*: ID, Nombre del ítem, Cantidad, status_id, event_id.

- TicketBooths
    - *Descripción*: Almacena información sobre las taquillas de los eventos, incluyendo su ubicación y el empleado encargado.
    - *Atributos*: ID, event_id, Ubicación, Número de contacto, manager_id.

- Visitors
    - *Descripción*: Registra la información de los visitantes que asisten a los eventos de anime.
    - *Atributos*: ID, Nombre, Documento de identificación, Género, Fecha de nacimiento, Email, Teléfono.

- TicketStatuses
    - *Descripción*: Gestiona los posibles estados de los tickets emitidos para los eventos (por ejemplo, pagado o reservado).
    - *Atributos*: ID, Estado del ticket.

- Orders

  - *Descripción*: Gestiona los pedidos realizados en las tiendas y restaurantes durante los eventos, registrando todos los detalles del pedido.
  - *Atributos*: ID, visitor_id, shop_id, cash_register_id, cashier_id, Valor total, Estado del pedido.

- Judges

  - *Descripción*: Almacena información sobre los jueces que participan en los eventos, evaluando concursos como el de cosplay.
  - *Atributos*: ID, Nombre, event_id.

- CosplayScores

  - *Descripción*: Registra los puntajes que los jueces asignan a los participantes de los concursos de cosplay.
  - *Atributos*: ID, participant_id, judge_id, Puntaje.

- TriviaQuestions

  - *Descripción*: Almacena las preguntas utilizadas en los concursos de trivia, categorizadas por tema y nivel de dificultad.
  - *Atributos*: ID, Pregunta, Respuesta correcta, Categoría, Dificultad, event_id, category_id.

- Discounts

  - *Descripción*: Gestiona los descuentos y promociones aplicados a productos o servicios durante los eventos.
  - *Atributos*: ID, shop_id, product_id, Tipo de descuento, Valor del descuento, Fecha de inicio, Fecha de fin.

- TriviaRounds

  - *Descripción*: Gestiona las rondas de los concursos de trivia, registrando a los participantes y ganadores de cada ronda.
  - *Atributos*: ID, event_id, Número de la ronda, participant1_id, participant2_id, winner_id.

- CosplayCategories

  - *Descripción*: Gestiona las categorías de los concursos de cosplay, permitiendo la clasificación de los participantes en diferentes grupos.
  - *Atributos*: ID, Nombre de la categoría, Descripción.

- OrderDetails
    - *Descripción*: Gestiona los detalles de los pedidos realizados en tiendas o restaurantes durante los eventos, registrando cada producto solicitado.
    - *Atributos*: ID, order_id, product_id, Cantidad, Precio unitario.

- Activities
    - *Descripción*: Registra información sobre las actividades que se realizan dentro de los eventos, como concursos de trivia, cosplay o demostraciones.
    - *Atributos*: ID, Nombre de la actividad, Descripción, Duración.

- ActivityParticipants
    - *Descripción*: Almacena los datos de los participantes en las actividades del evento, como concursos de trivia o cosplay.
    - *Atributos*: ID, visitor_id, activity_id, Fecha de inscripción.

- GeneralCategories
    - *Descripción*: Clasifica las diferentes categorías generales utilizadas para organizar actividades y preguntas de trivia.
    - *Atributos*: ID, Nombre de la categoría, Descripción.

- CashRegisters
    - *Descripción*: Almacena información sobre las cajas registradoras utilizadas en las tiendas y restaurantes durante los eventos.
    - *Atributos*: ID, Número de caja, shop_id, Ubicación.

- Products
    - *Descripción*: Gestiona la información de los productos disponibles en las tiendas y restaurantes durante los eventos.
    - *Atributos*: ID, Nombre del producto, Descripción, Precio, shop_id.

- EventSchedules
    - *Descripción*: Define los horarios de las actividades y eventos, permitiendo una organización temporal adecuada.
    - *Atributos*: ID, event_id, Fecha, Hora de inicio, Hora de fin.

- ProductCategories
    - *Descripción*: Organiza los productos en diferentes categorías para facilitar la gestión de los inventarios y la oferta en los eventos.

- *Atributos*: ID, Nombre de la categoría, Descripción.

- ActivityResults
  - *Descripción*: Registra los resultados de las actividades realizadas durante los eventos, como ganadores y puntajes.
  - *Atributos*: ID, activity_id, participant_id, Resultado, Puntaje.

- EmployeesStatus
  - *Descripción*: Define los posibles estados en los que pueden estar los empleados, como activo o inactivo.
  - *Atributos*: ID, Nombre del estado, Descripción, employees_id.

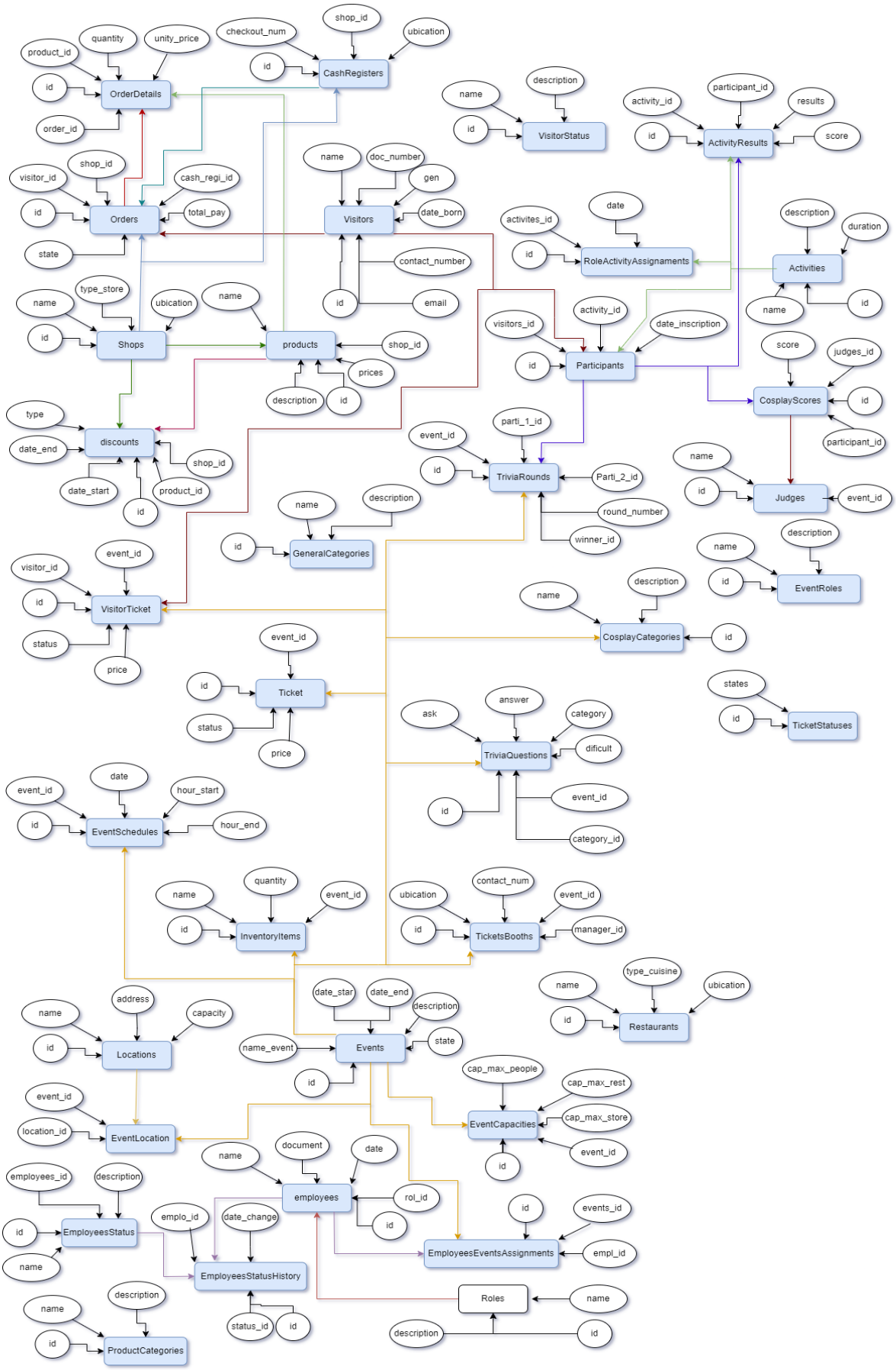
- VisitorStatus
  - *Descripción*: Almacena los diferentes estados posibles de los visitantes en relación con los eventos, como registrado o pendiente.
  - *Atributos*: ID, Nombre del estado, Descripción.

- Tickets
  - *Descripción*: Almacena la información relacionada con los tickets emitidos para los eventos, incluyendo precios y estados de pago.
  - *Atributos*: ID, visitor_id, event_id, Precio, Estado del ticket.

- EmployeeRoles
  - *Descripción*: Define los roles específicos de los empleados dentro de la organización de los eventos, como administrador o cajero.
  - *Atributos*: ID, Nombre del rol, Descripción.

- VisitorTickets
  - *Descripción*: Relaciona a los visitantes con los tickets comprados para asistir a los eventos.
  - *Atributos*: ID, visitor_id, ticket_id, Estado de la compra.

- EventRoles
  - *Descripción*: Almacena los roles asignados a las personas que organizan o participan en la logística de los eventos, como organizador o coordinador.
  - *Atributos*: ID, Nombre del rol, Descripción.


shop_id
ubication
checkout_num
quantity
unity_price
product_id
id
CashRegisters
OrderDetails
order_id
name
description
id
VisitorStatus
activity_id
participant_id
results
id
ActivityResults
score
shop_id
visitor_id
cash_regi_id
id
Orders
total_pay
state
name
doc_number
gen
Visitors
date_born
contact_number
id
email
activites_id
date
id
RoleActivityAssignaments
description
duration
Activities
name
id
type_store
name
ubication
name
id
Shops
products
shop_id
prices
description
id
visitors_id
activity_id
date_inscription
id
Participants
score
judges_id
CosplayScores
id
participant_id
type
date_end
discounts
shop_id
date_start
product_id
id
event_id
id
TriviaRounds
parti_1_id
Parti_2_id
round_number
winner_id
name
id
Judges
event_id
name
description
id
GeneralCategories
description
name
id
EventRoles
visitor_id
event_id
id
VisitorTicket
status
price
name
description
CosplayCategories
id
event_id
id
Ticket
status
price
states
id
TicketStatuses
ask
answer
category
dificult
TriviaQuestions
id
event_id
category_id
event_id
date
hour_start
id
EventSchedules
hour_end
name
quantity
event_id
id
InventoryItems
ubication
contact_num
event_id
id
TicketsBooths
manager_id
name
type_cuisine
ubication
id
Restaurants
name
address
capacity
id
Locations
date_star
date_end
description
name_event
Events
state
id
cap_max_people
cap_max_rest
cap_max_store
event_id
id
EventCapacities
event_id
location_id
EventLocation
name
document
date
employees
rol_id
id
employees_id
description
id
EmployeesStatus
name
emplo_id
date_change
EmployeesStatusHistory
status_id
id
id
events_id
empl_id
EmployeesEventsAssignments
Roles
name
description
id
description
name
id
ProductCategories

## Construcción del modelo lógico

El modelo lógico de la base de datos de gestión de eventos proporciona una representación detallada de la estructura utilizada para almacenar información relacionada con la organización y ejecución de diversos eventos, actividades y comercios. Este modelo define explícitamente las relaciones entre las tablas y los tipos de datos de cada atributo. A continuación, se presenta una descripción detallada de las tablas y sus relaciones:

1. **Locations**
    - Descripción: Almacena información detallada sobre las ubicaciones físicas donde se realizan los eventos.
    - Atributos:
        - id: Identificador único de la ubicación.
        - country: País donde se encuentra la ubicación.
        - city: Ciudad donde se encuentra la ubicación.
        - address: Dirección específica de la ubicación.
2. **AgeClassifications**
    - Descripción: Gestiona las clasificaciones de edad para los eventos y boletos.
    - Atributos:
        - id: Identificador único de la clasificación.
        - classification: Descripción de la clasificación de edad.
3. **EmployeeStatuses**
    - Descripción: Almacena los posibles estados laborales de los empleados.
    - Atributos:
        - id: Identificador único del estado.
        - status_name: Nombre del estado laboral.

4. **InventoryItemStatuses**
    - Descripción: Registra los diferentes estados de los ítems en inventario.
    - Atributos:
        - id: Identificador único del estado.
        - status_name: Nombre del estado del ítem.
5. **GeneralCategories**
    - Descripción: Almacena categorías generales para clasificar eventos, platos, tiendas y actividades.
    - Atributos:
        - id: Identificador único de la categoría.
        - category_name: Nombre de la categoría.
        - type: Tipo de categoría (EVENTO, PLATO, TIENDA, ACTIVIDAD).
6. **Roles**
    - Descripción: Gestiona los roles que pueden ser asignados a los empleados.
    - Atributos:
        - id: Identificador único del rol.
        - name: Nombre del rol.
7. **Activities**
    - Descripción: Almacena información sobre las actividades que se pueden realizar en los eventos.
    - Atributos:
        - id: Identificador único de la actividad.
        - name: Nombre de la actividad.

8. **DishTypes**
    - Descripción: Gestiona los diferentes tipos de platos ofrecidos en los eventos.
    - Atributos:
        - id: Identificador único del tipo de plato.
        - type_name: Nombre del tipo de plato.
9. **Organizers**
    - Descripción: Almacena información de los organizadores de los eventos.
    - Atributos:
        - id: Identificador único del organizador.
        - name: Nombre del organizador.
        - contact_info: Información de contacto del organizador.
10. **Events**
    - Descripción: Almacena datos esenciales de los eventos organizados.
    - Atributos:
        - id: Identificador único del evento.
        - name: Nombre del evento.
        - date_time: Fecha y hora de realización del evento.
        - organizer_id: Identificador del organizador asociado.
        - age_classification_id: Identificador de la clasificación de edad aplicable.
        - status: Estado actual del evento (ACTIVO, FINALIZADO, PENDIENTE).

11. **EventLocations**
    - Descripción: Relaciona eventos con sus ubicaciones específicas.
    - Atributos:
        - event_id: Identificador del evento.
        - location_id: Identificador de la ubicación.
12. **EventCapacities**
    - Descripción: Almacena la capacidad máxima permitida para personas, tiendas y restaurantes en un evento.
    - Atributos:
        - id: Identificador único de la capacidad.
        - event_id: Identificador del evento asociado.
        - max_capacity_people: Capacidad máxima de personas.
        - max_capacity_shops: Capacidad máxima de tiendas.
        - max_capacity_restaurants: Capacidad máxima de restaurantes.
13. **RoleActivityAssignments**
    - Descripción: Asigna roles específicos a las actividades dentro de un evento.
    - Atributos:
        - role_id: Identificador del rol.
        - activity_id: Identificador de la actividad.
        - assigned_date: Fecha de la asignación.

14. **Employees**

- Descripción: Almacena información sobre los empleados involucrados en la organización y ejecución de eventos.
- Atributos:
  - id: Identificador único del empleado.
  - name: Nombre del empleado.
  - identification: Documento de identidad del empleado.
  - birth_date: Fecha de nacimiento del empleado.
  - role_id: Identificador del rol asignado.
  - status_id: Identificador del estado laboral actual.

15. **EmployeeEventAssignments**

- Descripción: Asigna empleados a eventos específicos para realizar tareas designadas.
- Atributos:
  - id: Identificador único de la asignación.
  - employee_id: Identificador del empleado asignado.
  - event_id: Identificador del evento asociado.

16. **EmployeeStatusHistory**

- Descripción: Registra los cambios en el estado laboral de los empleados a lo largo del tiempo.
- Atributos:
  - id: Identificador único del registro histórico.
  - employee_id: Identificador del empleado.
  - status_id: Identificador del nuevo estado laboral.

        - change_date: Fecha del cambio de estado.

17. **InventoryItems**
    - Descripción: Gestiona el inventario de artículos utilizados en los eventos.
    - Atributos:
        - id: Identificador único del artículo.
        - name: Nombre del artículo.
        - quantity: Cantidad disponible del artículo.
        - status_id: Identificador del estado del artículo.
        - event_id: Identificador del evento asociado.

18. **TicketBooths**
    - Descripción: Administra las taquillas donde se venden los boletos para los eventos.
    - Atributos:
        - id: Identificador único de la taquilla.
        - event_id: Identificador del evento donde se ubica la taquilla.
        - location: Ubicación específica de la taquilla dentro del evento.
        - contact_number: Número de contacto para información y ventas.
        - manager_id: Identificador del empleado que gestiona la taquilla.

19. **Visitors**

    - Descripción: Registra la información de los visitantes que asisten a los eventos.
    - Atributos:
        - id: Identificador único del visitante.
        - name: Nombre del visitante.
        - identification_document: Documento de identificación del visitante.
        - gender: Género del visitante.
        - birth_date: Fecha de nacimiento del visitante.
        - email: Correo electrónico del visitante.
        - phone_number: Número de teléfono del visitante.

20. **TicketStatuses**

    - Descripción: Define los posibles estados de un boleto (pagado, reservado).
    - Atributos:
        - id: Identificador único del estado del boleto.
        - status: Estado del boleto.

21. **Tickets**

- Descripción: Gestiona la emisión y el control de los boletos para los eventos.
- Atributos:
    - id: Identificador único del boleto.
    - name: Nombre descriptivo del boleto.
    - price: Precio del boleto.
    - age_classification_id: Identificador de la clasificación de edad aplicable.

    - additional_cost: Costos adicionales que pueden aplicar para actividades específicas.
    - status_id: Identificador del estado del boleto.
    - ticket_booth_id: Identificador de la taquilla donde se vende el boleto.
    - customer_id: Identificador del visitante que compra el boleto.

22. **EventActivities**

- Descripción: Gestiona las actividades que se llevan a cabo durante los eventos.
- Atributos:
    - id: Identificador único de la actividad.
    - name: Nombre de la actividad.
    - type: Tipo de actividad (cosplay, trivia).
    - number_of_participants: Número máximo de participantes permitidos.
    - event_id: Identificador del evento asociado.
    - start_time: Hora de inicio de la actividad.
    - manager_id: Identificador del empleado responsable de la actividad.
    - category_id: Identificador de la categoría general a la que pertenece la actividad.

23. **ActivityParticipants**

- Descripción: Registra la participación de los visitantes en las diversas actividades de los eventos.
- Atributos:
    - id: Identificador único del registro de participación.
    - activity_id: Identificador de la actividad en la que participa.
    - visitor_id: Identificador del visitante participante.

- is_winner: Indicador de si el participante fue ganador en la actividad.

24. **Prizes**

- Descripción: Gestiona los premios entregados en las actividades de los eventos.
- Atributos:
  - id: Identificador único del premio.
  - type: Tipo de premio.
  - description: Descripción detallada del premio.
  - value: Valor monetario del premio.
  - status: Estado del premio (disponible, entregado).
  - activity_id: Identificador de la actividad asociada.
  - participant_id: Identificador del participante ganador.

25. **Shops**

- Descripción: Administra las tiendas y restaurantes disponibles en los eventos.
- Atributos:
  - id: Identificador único de la tienda o restaurante.
  - name: Nombre del establecimiento.
  - category: Categoría del establecimiento (tienda, restaurante).
  - event_id: Identificador del evento donde se ubica.
  - manager_id: Identificador del empleado encargado del establecimiento.

26. **Products**

- Descripción: Gestiona los productos disponibles para la venta en las tiendas de los eventos.
- Atributos:

- id: Identificador único del producto.
- shop_id: Identificador de la tienda que ofrece el producto.
- name: Nombre del producto.
- description: Descripción del producto.
- manufacturer: Fabricante del producto.
- type: Tipo de producto.
- available_quantity: Cantidad disponible para la venta.
- price: Precio del producto.

27. **Dishes**

- Descripción: Controla los platos ofrecidos en los restaurantes de los eventos.
- Atributos:
  - id: Identificador único del plato.
  - restaurant_id: Identificador del restaurante que ofrece el plato.
  - name: Nombre del plato.
  - price: Precio del plato.
  - type_id: Identificador del tipo de plato.
  - preparation_time: Tiempo estimado de preparación.

28. **Ingredients**

- Descripción: Almacena información sobre los ingredientes utilizados en la preparación de platos en los restaurantes.
- Atributos:
  - id: Identificador único del ingrediente.
  - name: Nombre del ingrediente.

- available_quantity: Cantidad disponible del ingrediente.

29. **DishIngredients**

- Descripción: Relaciona los platos con los ingredientes necesarios para su preparación.
- Atributos:
  - dish_id: Identificador del plato.
  - ingredient_id: Identificador del ingrediente.
  - quantity_required: Cantidad del ingrediente requerida para el plato.

30. **CashRegisters**

- Descripción: Administra las cajas registradoras utilizadas en los comercios de los eventos.
- Atributos:
  - id: Identificador único de la caja registradora.
  - status: Estado operativo de la caja (activo, inactivo).
  - operator_id: Identificador del empleado que opera la caja.
  - opening_amount: Monto inicial en la caja al abrir.
  - closing_amount: Monto final en la caja al cerrar.

31. **Orders**

- Descripción: Gestiona los pedidos realizados en las tiendas y restaurantes durante los eventos.
- Atributos:
  - id: Identificador único del pedido.
  - visitor_id: Identificador del visitante que realiza el pedido.
  - shop_id: Identificador de la tienda o restaurante.

    - cash_register_id: Identificador de la caja registradora utilizada para el pago.
    - cashier_id: Identificador del cajero que procesa el pedido.
    - total_value: Valor total del pedido.
    - status: Estado del pedido (registrado, pagado, entregado).

32. **Judges**

- Descripción: Almacena información sobre los jueces involucrados en la evaluación de actividades y competencias en los eventos.
- Atributos:
    - id: Identificador único del juez.
    - name: Nombre del juez.
    - event_id: Identificador del evento donde el juez participa.

33. **CosplayScores**

- Descripción: Registra los puntajes asignados a los participantes en concursos de cosplay.
- Atributos:
    - id: Identificador único del puntaje.
    - participant_id: Identificador del participante.
    - judge_id: Identificador del juez que asigna el puntaje.
    - score: Puntaje dado al participante.

34. **TriviaQuestions**

- Descripción: Administra las preguntas utilizadas en los concursos de trivia durante los eventos.
- Atributos:
    - id: Identificador único de la pregunta.

- question: Texto de la pregunta.
- correct_answer: Respuesta correcta a la pregunta.
- category: Categoría de la pregunta.
- difficulty: Nivel de dificultad de la pregunta (fácil, intermedio, difícil).
- event_id: Identificador del evento asociado.
- category_id: Identificador de la categoría general relacionada.

35. **Discounts**

- Descripción: Gestiona descuentos y promociones ofrecidas en las tiendas de los eventos.
- Atributos:
  - id: Identificador único del descuento.
  - shop_id: Identificador de la tienda que ofrece el descuento.
  - product_id: Identificador del producto al que aplica el descuento.
  - discount_type: Tipo de descuento (porcentaje, cantidad fija, paquete).
  - discount_value: Valor del descuento aplicado.
  - start_date: Fecha de inicio del descuento.
  - end_date: Fecha de finalización del descuento.

36. **TriviaRounds**

- Descripción: Gestiona las rondas de los concursos de trivia en los eventos.
- Atributos:
  - id: Identificador único de la ronda.
  - event_id: Identificador del evento asociado.
  - round_number: Número de la ronda dentro del concurso.

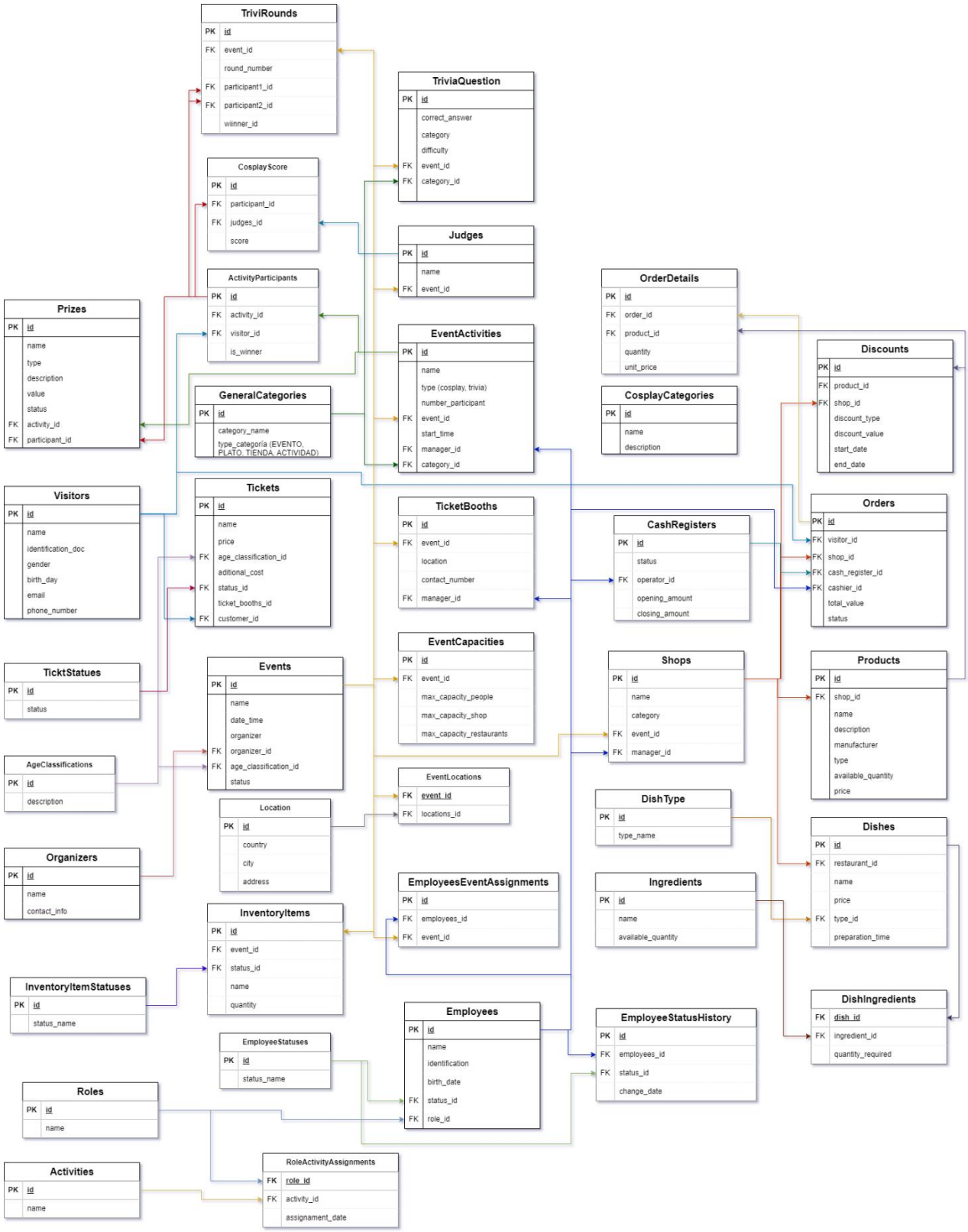
- participant1_id: Identificador del primer participante.
- participant2_id: Identificador del segundo participante.
- winner_id: Identificador del ganador de la ronda.

37. **CosplayCategories**

- Descripción: Define las diferentes categorías para los concursos de cosplay.
- Atributos:
  - id: Identificador único de la categoría.
  - name: Nombre de la categoría.
  - description: Descripción de la categoría.

38. **OrderDetails**

- Descripción: Registra los detalles de los pedidos realizados en tiendas o restaurantes.
- Atributos:
  - id: Identificador único del detalle del pedido.
  - order_id: Identificador del pedido asociado.
  - product_id: Identificador del producto ordenado.
  - quantity: Cantidad del producto ordenado.
  - unit_price: Precio unitario del producto.


TriviRounds
PK id
FK event_id
round_number
FK participant1_id
FK participant2_id
winner_id
TriviaQuestion
PK id
correct_answer
category
difficulty
FK event_id
FK category_id
CosplayScore
PK id
FK participant_id
FK judges_id
score
Judges
PK id
name
FK event_id
ActivityParticipants
PK id
FK activity_id
FK visitor_id
is_winner
OrderDetails
PK id
FK order_id
FK product_id
quantity
unit_price
Prizes
PK id
name
type
description
value
status
FK activity_id
FK participant_id
EventActivities
PK id
name
type (cosplay, trivia)
number_participant
FK event_id
start_time
FK manager_id
FK category_id
Discounts
PK id
FK product_id
FK shop_id
discount_type
discount_value
start_date
end_date
GeneralCategories
PK id
category_name
type_categoria (EVENTO, PLATO, TIENDA, ACTIVIDAD)
CosplayCategories
PK id
name
description
Visitors
PK id
name
identification_doc
gender
birth_day
email
phone_number
Tickets
PK id
name
price
FK age_classification_id
aditional_cost
FK status_id
ticket_booths_id
FK customer_id
TicketBooths
PK id
FK event_id
location
contact_number
FK manager_id
CashRegisters
PK id
status
FK operator_id
opening_amount
closing_amount
Orders
PK id
FK visitor_id
FK shop_id
FK cash_register_id
FK cashier_id
total_value
status
EventCapacities
PK id
FK event_id
max_capacity_people
max_capacity_shop
max_capacity_restaurants
TicktStatues
PK id
status
Events
PK id
name
date_time
organizer
FK organizer_id
FK age_classification_id
status
Shops
PK id
name
category
FK event_id
FK manager_id
Products
PK id
FK shop_id
name
description
manufacturer
type
available_quantity
price
AgeClassifications
PK id
description
EventLocations
FK event_id
FK locations_id
DishType
PK id
type_name
Location
PK id
country
city
address
Dishes
PK id
FK restaurant_id
name
price
FK type_id
preparation_time
Organizers
PK id
name
contact_info
EmployeesEventAssignments
PK id
FK employees_id
FK event_id
Ingredients
PK id
name
available_quantity
InventoryItems
PK id
FK event_id
FK status_id
name
quantity
InventoryItemStatuses
PK id
status_name
Employees
PK id
name
identification
birth_date
FK status_id
FK role_id
EmployeeStatusHistory
PK id
FK employees_id
FK status_id
change_date
DishIngredients
FK dish_id
FK ingredient_id
quantity_required
EmployeeStatuses
PK id
status_name
Roles
PK id
name
Activities
PK id
name
RoleActivityAssignments
FK role_id
FK activity_id
assignament_date

## Normalización del modelo lógico

### Primera forma normal (1NF)

Para que una tabla esté en la Primera Forma Normal (1FN), debe cumplir con los siguientes requisitos: cada columna debe contener solo valores atómicos y cada fila debe ser única. Esto significa que no deben existir grupos de valores repetitivos o múltiples valores en una sola columna.

En el caso de tu proyecto de gestión de eventos, parece que todas las tablas cumplen con la 1FN. Aquí algunos ejemplos:

1. **Locations**
    - Descripción: Esta tabla almacena información detallada sobre las ubicaciones físicas de los eventos. Las columnas incluyen id, country, city, y address, cada una conteniendo un único valor atómico por registro. No hay grupos repetitivos ni múltiples valores en una sola columna.
2. **Employees**
    - Descripción: Registra información sobre los empleados involucrados en la organización y ejecución de eventos. Incluye columnas como id, name, identification, birth_date, role_id, y status_id, con cada columna conteniendo un solo valor atómico.
3. **Events**
    - Descripción: Almacena datos esenciales de los eventos organizados. Tiene columnas como id, name, date_time, organizer_id, age_classification_id, y status. Cada columna es atómica, asegurando que no haya múltiples valores en una sola columna y cada evento es identificable de manera única.
4. **Tickets**
    - Descripción: Gestiona la emisión y el control de los boletos para los eventos. Las columnas id, name, price, age_classification_id, additional_cost,

status_id, ticket_booth_id, y customer_id contienen valores atómicos y cada fila es única, cumpliendo con la 1FN.

5. **Dishes**
    - Descripción: Controla los platos ofrecidos en los restaurantes de los eventos. Con columnas como id, restaurant_id, name, price, type_id, y preparation_time, cada una conteniendo un solo valor atómico.

**Segunda forma normal (2FN)**

La Segunda Forma Normal (2FN) se alcanza en una base de datos cuando todas las tablas están en Primera Forma Normal y además, todos los atributos que no son parte de la clave primaria dependen únicamente de toda la clave primaria y no de una parte de ella. Esto significa que no debería haber dependencias parciales de un atributo en una parte de la clave primaria en tablas con claves primarias compuestas.

Para verificar si las tablas están en Segunda Forma Normal, es necesario revisar las tablas con claves primarias compuestas para asegurarnos de que no haya atributos que dependan solo de una parte de la clave primaria. A continuación, analizo cada tabla relevante de tu base de datos bajo esta norma:

1. **EventLocations**
    - Clave Primaria: (event_id, location_id)
    - Atributos: No hay atributos adicionales dependiendo de partes de la clave primaria.
    - **Cumple 2FN**: Sí.
2. **RoleActivityAssignments**
    - Clave Primaria: (role_id, activity_id)
    - Atributos: assigned_date depende de la combinación de role_id y activity_id, no de uno solo.

- **Cumple 2FN**: Sí.

3. **EmployeeEventAssignments**
    - Clave Primaria: id (único, no compuesto)
    - **Cumple 2FN**: Sí.
4. **DishIngredients**
    - Clave Primaria: (dish_id, ingredient_id)
    - Atributos: quantity_required depende de la combinación completa de dish_id y ingredient_id.
    - **Cumple 2FN**: Sí.
5. **TriviaRounds**
    - Clave Primaria: id (único, no compuesto)
    - **Cumple 2FN**: Sí.
6. **OrderDetails**
    - Clave Primaria: id (único, no compuesto)
    - **Cumple 2FN**: Sí.

Todas las demás tablas tienen claves primarias únicas, no compuestas, lo que las coloca automáticamente en Segunda Forma Normal, ya que no puede haber dependencia parcial si solo hay un único atributo como clave primaria.

Por tanto, en base a esta revisión, todas las tablas mencionadas y el resto que tienen claves primarias únicas en tu base de datos están en Segunda Forma Normal. Esto asegura una mayor normalización y eficiencia en el manejo de los datos dentro de tu sistema de gestión de eventos.

**Tercera forma normal (3FN)**

La Tercera Forma Normal (3FN) se alcanza en una base de datos cuando está en Segunda Forma Normal y además, no existe ninguna dependencia transicional; es decir, ningún atributo no clave depende de otro atributo no clave. Esto ayuda a eliminar las redundancias no deseadas y garantiza que todos los atributos no clave dependan directamente de la clave primaria.

Vamos a revisar algunas de las tablas clave de tu base de datos para verificar si cumplen con la Tercera Forma Normal:

1. **Events**
    - Claves Primarias: id
    - Dependencias: organizer_id y age_classification_id dependen directamente de id y no de otros atributos no clave.
    - **Cumple 3FN**: Sí.
2. **Tickets**
    - Claves Primarias: id
    - Dependencias: age_classification_id, status_id, ticket_booth_id, customer_id dependen directamente de id.
    - **Cumple 3FN**: Sí.
3. **Employees**
    - Claves Primarias: id
    - Dependencias: role_id y status_id dependen directamente de id.
    - **Cumple 3FN**: Sí.
4. **EventActivities**
    - Claves Primarias: id

- Dependencias: event_id, manager_id, category_id dependen directamente de id y no de otros atributos no clave.
- **Cumple 3FN**: Sí.

5. **Shops**
   - Claves Primarias: id
   - Dependencias: event_id y manager_id dependen directamente de id.
   - **Cumple 3FN**: Sí.
6. **Orders**
   - Claves Primarias: id
   - Dependencias: visitor_id, shop_id, cash_register_id, cashier_id dependen directamente de id.
   - **Cumple 3FN**: Sí.
7. **Products**
   - Claves Primarias: id
   - Dependencias: shop_id depende directamente de id.
   - **Cumple 3FN**: Sí.
8. **Dishes**
   - Claves Primarias: id
   - Dependencias: restaurant_id y type_id dependen directamente de id.
   - **Cumple 3FN**: Sí.

Estas tablas, junto con las otras en tu base de datos que tienen dependencias directas y únicas de sus claves primarias a sus atributos no clave, están en Tercera Forma Normal. Esto asegura que la información está organizada de manera óptima, eliminando redundancias y mejorando la integridad y eficiencia de los datos en tu sistema de gestión de eventos.

## Cuarta forma normal (4FN)

La Cuarta Forma Normal (4FN) se aplica a situaciones en bases de datos donde existen múltiples relaciones independientes entre entidades, y esencialmente elimina las dependencias multivaluadas. Para que una tabla esté en 4FN, debe estar en la Tercera Forma Normal y además no tener dependencias multivaluadas, es decir, ningún atributo no clave debe tener más de un valor independiente y no relacionado para un único valor de clave primaria.

Para analizar si tus tablas están en Cuarta Forma Normal, examinaremos si hay alguna dependencia multivaluada en las tablas con claves primarias y atributos que puedan tener múltiples valores independientes para una sola clave primaria. Veamos algunos ejemplos:

1. **EventLocations**
    - **Dependencias**: La tabla vincula eventos con ubicaciones mediante las claves event_id y location_id.
    - **4FN**: Sí, porque cada combinación de evento y ubicación es única y no implica dependencias multivaluadas entre otros atributos.
2. **RoleActivityAssignments**
    - **Dependencias**: Asigna roles a actividades en fechas específicas.
    - **4FN**: Sí, porque cada combinación de rol y actividad es única y no tiene múltiples valores independientes para un único valor de clave primaria.
3. **EmployeeEventAssignments**
    - **Dependencias**: Vincula empleados con eventos.
    - **4FN**: Sí, porque cada asignación de empleado a evento es única y no involucra múltiples roles o estados independientes para una misma clave.
4. **ActivityParticipants**
    - **Dependencias**: Relaciona participantes con actividades específicas.

- **4FN**: Sí, dado que cada participante está asociado a una actividad sin implicar múltiples participaciones independientes para una única clave primaria.

Las tablas revisadas y las demás tablas similares en tu base de datos parecen cumplir con la Cuarta Forma Normal, ya que no exhiben dependencias multivaluadas. Esto indica una estructura de datos eficiente y bien normalizada, donde las relaciones entre las entidades están claramente definidas sin redundancias ni ambigüedades.

## Construcción del modelo físico

El modelo físico para la gestión de eventos se construye a partir del modelo lógico, realizando una implementación específica en SQL que incluye la creación de tablas, asignación de tipos de datos a cada columna, y definición de restricciones y relaciones entre las tablas mediante claves foráneas. Por ejemplo, en las tablas, los identificadores como id se definen como INT AUTO_INCREMENT, mientras que nombres y direcciones se almacenan en campos VARCHAR de longitud adecuada. Se establecen claves primarias para garantizar la unicidad de los registros y claves foráneas para mantener la integridad referencial, como en la tabla de eventos donde organizer_id referencia a la tabla Organizers. Además, para mejorar el rendimiento de las consultas, se definen índices en columnas que se utilizan frecuentemente en los filtros y ordenaciones, asegurando así una gestión eficiente y óptima del rendimiento en operaciones y consultas de datos en un entorno de base de datos relacional.

**Código:**

```
-- Use the database
USE bborfnnyj5u5a3o2odu6;

-- Table for locations
CREATE TABLE Locations (
    id INT AUTO_INCREMENT PRIMARY KEY,
    country VARCHAR(100) NOT NULL,
    city VARCHAR(100) NOT NULL,
    address VARCHAR(255) NOT NULL,
    UNIQUE (country, city, address)
);

-- Table for age classifications
```

```sql
CREATE TABLE AgeClassifications (
    id INT AUTO_INCREMENT PRIMARY KEY,
    classification VARCHAR(10) NOT NULL UNIQUE
);

-- Table for employee statuses
CREATE TABLE EmployeeStatuses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    status_name VARCHAR(50) NOT NULL UNIQUE
);

-- Table for inventory item statuses
CREATE TABLE InventoryItemStatuses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    status_name VARCHAR(50) NOT NULL UNIQUE
);

-- Table for general categories
CREATE TABLE GeneralCategories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    category_name VARCHAR(100) NOT NULL,
    type ENUM('EVENT', 'DISH', 'SHOP', 'ACTIVITY') NOT NULL,
    UNIQUE (category_name, type)
);

-- Table for employee roles
CREATE TABLE Roles (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE
);

-- Table for activities
CREATE TABLE Activities (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE
);

-- Table for dish types
CREATE TABLE DishTypes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    type_name VARCHAR(100) NOT NULL UNIQUE
);

-- Table for organizers
CREATE TABLE Organizers (
```

```sql
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL UNIQUE,
    contact_info TEXT
);

-- Table for storing basic event information
CREATE TABLE Events (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    date_time DATETIME NOT NULL,
    organizer_id INT,
    age_classification_id INT,
    status ENUM('ACTIVE', 'FINISHED', 'PENDING') NOT NULL,
    UNIQUE (name, date_time),
    FOREIGN KEY (organizer_id) REFERENCES Organizers(id) ON DELETE SET NULL,
    FOREIGN KEY (age_classification_id) REFERENCES AgeClassifications(id) ON
DELETE SET NULL
);

-- Table to link events with locations
CREATE TABLE EventLocations (
    event_id INT,
    location_id INT,
    PRIMARY KEY (event_id, location_id),
    FOREIGN KEY (event_id) REFERENCES Events(id) ON DELETE CASCADE,
    FOREIGN KEY (location_id) REFERENCES Locations(id) ON DELETE CASCADE
);

-- Table to store event capacities
CREATE TABLE EventCapacities (
    id INT AUTO_INCREMENT PRIMARY KEY,
    event_id INT,
    max_capacity_people INT NOT NULL,
    max_capacity_shops INT NOT NULL,
    max_capacity_restaurants INT NOT NULL,
    FOREIGN KEY (event_id) REFERENCES Events(id) ON DELETE CASCADE
);

-- Table to assign activities to roles
CREATE TABLE RoleActivityAssignments (
    role_id INT,
    activity_id INT,
    assigned_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (role_id, activity_id),
    FOREIGN KEY (role_id) REFERENCES Roles(id) ON DELETE CASCADE,
```

```
    FOREIGN KEY (activity_id) REFERENCES Activities(id) ON DELETE CASCADE
);

-- Table for employees
CREATE TABLE Employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    identification VARCHAR(50) NOT NULL UNIQUE,
    birth_date DATE NOT NULL,
    role_id INT,
    status_id INT,
    FOREIGN KEY (role_id) REFERENCES Roles(id) ON DELETE SET NULL,
    FOREIGN KEY (status_id) REFERENCES EmployeeStatuses(id) ON DELETE SET
NULL
);

-- Table for employee assignment to events
CREATE TABLE EmployeeEventAssignments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employee_id INT,
    event_id INT,
    FOREIGN KEY (employee_id) REFERENCES Employees(id) ON DELETE CASCADE,
    FOREIGN KEY (event_id) REFERENCES Events(id) ON DELETE CASCADE
);

-- History table for employee status changes
CREATE TABLE EmployeeStatusHistory (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employee_id INT,
    status_id INT,
    change_date DATETIME NOT NULL,
    FOREIGN KEY (employee_id) REFERENCES Employees(id) ON DELETE CASCADE,
    FOREIGN KEY (status_id) REFERENCES EmployeeStatuses(id) ON DELETE
CASCADE
);

-- Table for inventory items
CREATE TABLE InventoryItems (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    quantity INT NOT NULL,
    status_id INT,
    event_id INT,
    FOREIGN KEY (status_id) REFERENCES InventoryItemStatuses(id) ON DELETE
SET NULL,
```

```
    FOREIGN KEY (event_id) REFERENCES Events(id) ON DELETE SET NULL
);

-- Table for ticket booths
CREATE TABLE TicketBooths (
    id INT AUTO_INCREMENT PRIMARY KEY,
    event_id INT,
    location VARCHAR(255) NOT NULL,
    contact_number VARCHAR(20),
    manager_id INT,
    FOREIGN KEY (event_id) REFERENCES Events(id) ON DELETE CASCADE,
    FOREIGN KEY (manager_id) REFERENCES Employees(id) ON DELETE SET NULL
);

-- Table for visitors
CREATE TABLE Visitors (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    identification_document VARCHAR(50) NOT NULL UNIQUE,
    gender ENUM('MALE', 'FEMALE', 'OTHER') NOT NULL,
    birth_date DATE NOT NULL,
    email VARCHAR(100),
    phone_number VARCHAR(20)
);

-- Table for ticket statuses
CREATE TABLE TicketStatuses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    status ENUM('PAID', 'RESERVED') NOT NULL
);

-- Table for tickets
CREATE TABLE Tickets (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    age_classification_id INT,
    additional_cost DECIMAL(10, 2) DEFAULT 0,
    status_id INT,
    ticket_booth_id INT,
    customer_id INT,
    FOREIGN KEY (age_classification_id) REFERENCES AgeClassifications(id) ON
DELETE SET NULL,
    FOREIGN KEY (status_id) REFERENCES TicketStatuses(id) ON DELETE SET
NULL,
```

```
    FOREIGN KEY (ticket_booth_id) REFERENCES TicketBooths(id) ON DELETE
CASCADE,
    FOREIGN KEY (customer_id) REFERENCES Visitors(id) ON DELETE CASCADE
);

-- Table for activities within an event
CREATE TABLE EventActivities (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    type ENUM('COSPLAY', 'TRIVIA') NOT NULL,
    number_of_participants INT NOT NULL,
    event_id INT,
    start_time DATETIME NOT NULL,
    manager_id INT,
    category_id INT,
    FOREIGN KEY (event_id) REFERENCES Events(id) ON DELETE CASCADE,
    FOREIGN KEY (manager_id) REFERENCES Employees(id) ON DELETE SET NULL,
    FOREIGN KEY (category_id) REFERENCES GeneralCategories(id) ON DELETE SET
NULL
);

-- Table for participants in activities
CREATE TABLE ActivityParticipants (
    id INT AUTO_INCREMENT PRIMARY KEY,
    activity_id INT,
    visitor_id INT,
    is_winner BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (activity_id) REFERENCES EventActivities(id) ON DELETE
CASCADE,
    FOREIGN KEY (visitor_id) REFERENCES Visitors(id) ON DELETE CASCADE
);

-- Table for prizes for activities
CREATE TABLE Prizes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    type VARCHAR(100) NOT NULL,
    description TEXT NOT NULL,
    value DECIMAL(10, 2) NOT NULL,
    status ENUM('AVAILABLE', 'DELIVERED') NOT NULL,
    activity_id INT,
    participant_id INT,
    FOREIGN KEY (activity_id) REFERENCES EventActivities(id) ON DELETE
CASCADE,
    FOREIGN KEY (participant_id) REFERENCES ActivityParticipants(id) ON
DELETE CASCADE
```

```
);

-- Table for shops and restaurants at the event
CREATE TABLE Shops (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    category ENUM('SHOP', 'RESTAURANT') NOT NULL,
    event_id INT,
    manager_id INT,
    FOREIGN KEY (event_id) REFERENCES Events(id) ON DELETE CASCADE,
    FOREIGN KEY (manager_id) REFERENCES Employees(id) ON DELETE SET NULL
);

-- Table for products at the shops
CREATE TABLE Products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    shop_id INT,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    manufacturer VARCHAR(100),
    type VARCHAR(50),
    available_quantity INT NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (shop_id) REFERENCES Shops(id) ON DELETE CASCADE
);

-- Table for dishes at restaurants
CREATE TABLE Dishes (
    id INT AUTO_INCREMENT PRIMARY KEY,
    restaurant_id INT,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    type_id INT,
    preparation_time INT NOT NULL,
    FOREIGN KEY (restaurant_id) REFERENCES Shops(id) ON DELETE CASCADE,
    FOREIGN KEY (type_id) REFERENCES DishTypes(id) ON DELETE SET NULL
);

-- Table for ingredients
CREATE TABLE Ingredients (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    available_quantity INT NOT NULL
);
```

```sql
-- Table for linking dishes with required ingredients
CREATE TABLE DishIngredients (
    dish_id INT,
    ingredient_id INT,
    quantity_required INT NOT NULL,
    PRIMARY KEY (dish_id, ingredient_id),
    FOREIGN KEY (dish_id) REFERENCES Dishes(id) ON DELETE CASCADE,
    FOREIGN KEY (ingredient_id) REFERENCES Ingredients(id) ON DELETE CASCADE
);

-- Table for cash registers
CREATE TABLE CashRegisters (
    id INT AUTO_INCREMENT PRIMARY KEY,
    status ENUM('ACTIVE', 'INACTIVE') DEFAULT 'INACTIVE',
    operator_id INT,
    opening_amount DECIMAL(10, 2),
    closing_amount DECIMAL(10, 2),
    FOREIGN KEY (operator_id) REFERENCES Employees(id) ON DELETE SET NULL
);

-- Table for orders
CREATE TABLE Orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    visitor_id INT,
    shop_id INT,
    cash_register_id INT,
    cashier_id INT,
    total_value DECIMAL(10, 2) NOT NULL,
    status ENUM('REGISTERED', 'PAID', 'DELIVERED') NOT NULL,
    FOREIGN KEY (visitor_id) REFERENCES Visitors(id) ON DELETE CASCADE,
    FOREIGN KEY (shop_id) REFERENCES Shops(id) ON DELETE CASCADE,
    FOREIGN KEY (cash_register_id) REFERENCES CashRegisters(id) ON DELETE
CASCADE,
    FOREIGN KEY (cashier_id) REFERENCES Employees(id) ON DELETE SET NULL
);

-- Table for judges at events
CREATE TABLE Judges (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    event_id INT,
    FOREIGN KEY (event_id) REFERENCES Events(id) ON DELETE CASCADE
);

-- Table for Cosplay scores
```

```sql
CREATE TABLE CosplayScores (
    id INT AUTO_INCREMENT PRIMARY KEY,
    participant_id INT,
    judge_id INT,
    score DECIMAL(4, 2) NOT NULL,
    FOREIGN KEY (participant_id) REFERENCES ActivityParticipants(id) ON
DELETE CASCADE,
    FOREIGN KEY (judge_id) REFERENCES Judges(id) ON DELETE CASCADE
);

-- Table for trivia questions
CREATE TABLE TriviaQuestions (
    id INT AUTO_INCREMENT PRIMARY KEY,
    question TEXT NOT NULL,
    correct_answer TEXT NOT NULL,
    category VARCHAR(100) NOT NULL,
    difficulty ENUM('EASY', 'INTERMEDIATE', 'HARD') NOT NULL,
    event_id INT,
    category_id INT,
    FOREIGN KEY (event_id) REFERENCES Events(id) ON DELETE CASCADE,
    FOREIGN KEY (category_id) REFERENCES GeneralCategories(id) ON DELETE SET
NULL
);

-- Table for discounts
CREATE TABLE Discounts (
    id INT AUTO_INCREMENT PRIMARY KEY,
    shop_id INT,
    product_id INT,
    discount_type ENUM('PERCENTAGE', 'FIXED_AMOUNT', 'BUNDLE') NOT NULL,
    discount_value DECIMAL(10, 2) NOT NULL,
    start_date DATETIME NOT NULL,
    end_date DATETIME NOT NULL,
    FOREIGN KEY (shop_id) REFERENCES Shops(id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES Products(id) ON DELETE CASCADE
);

-- Table for trivia rounds
CREATE TABLE TriviaRounds (
    id INT AUTO_INCREMENT PRIMARY KEY,
    event_id INT,
    round_number INT NOT NULL,
    participant1_id INT,
    participant2_id INT,
    winner_id INT,
```

```
    FOREIGN KEY (event_id) REFERENCES Events(id) ON DELETE CASCADE,
    FOREIGN KEY (participant1_id) REFERENCES ActivityParticipants(id) ON
DELETE CASCADE,
    FOREIGN KEY (participant2_id) REFERENCES ActivityParticipants(id) ON
DELETE CASCADE,
    FOREIGN KEY (winner_id) REFERENCES ActivityParticipants(id) ON DELETE
SET NULL
);

-- Table for cosplay categories
CREATE TABLE CosplayCategories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    description TEXT
);

-- Table for order details
CREATE TABLE OrderDetails (
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT NOT NULL,
    unit_price DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (order_id) REFERENCES Orders(id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES Products(id) ON DELETE CASCADE
);
```

## Diagrama E-R

El diagrama Entidad-Relación (ER) para nuestra base de datos de gestión de eventos es una herramienta visual esencial que ilustra cómo se relacionan las diferentes entidades dentro del sistema. Este diagrama facilita la comprensión y comunicación de la estructura de la base de datos de manera efectiva y eficiente.

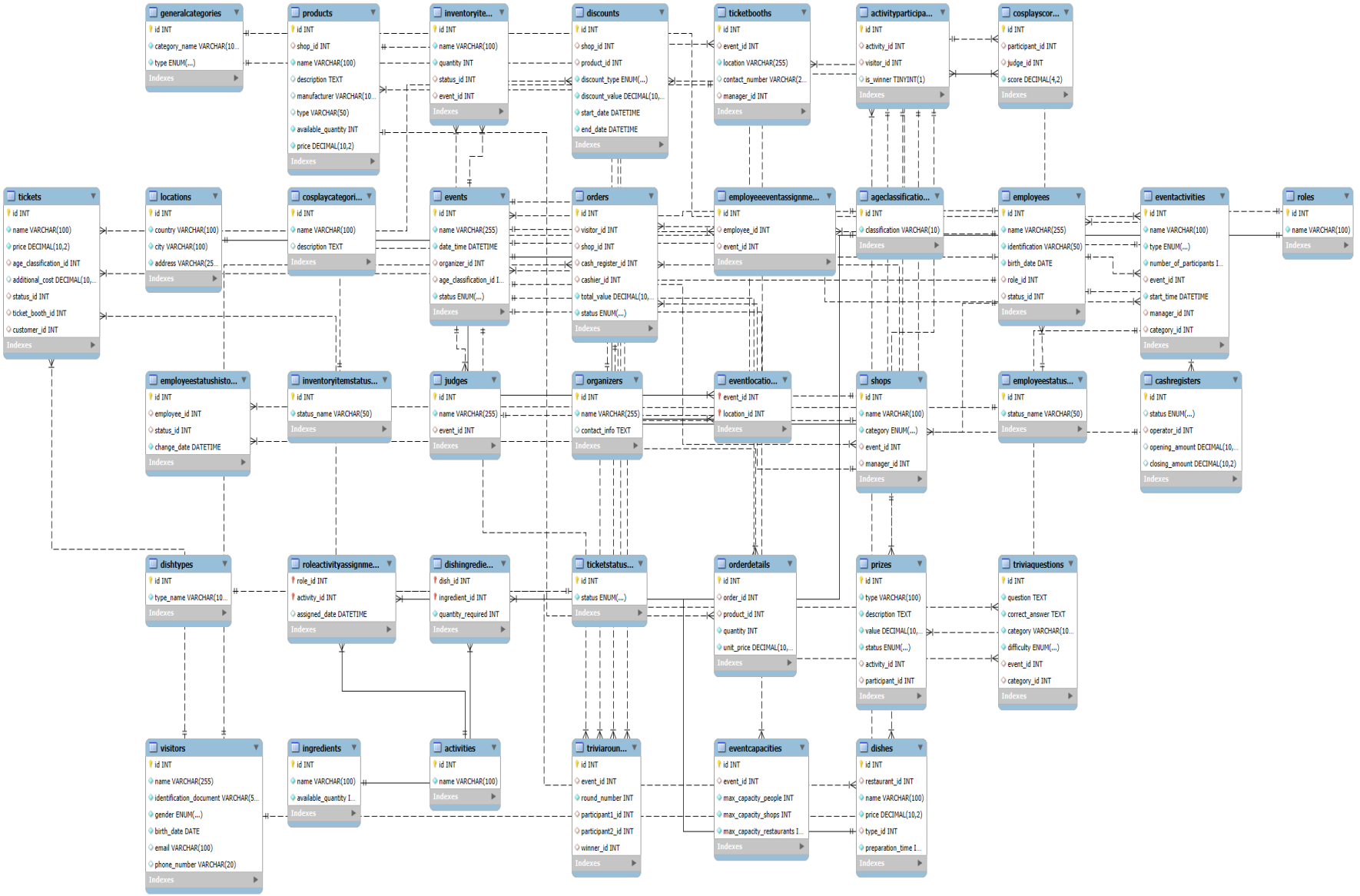
En nuestro diagrama ER, las principales entidades incluyen **Eventos**, **Visitantes**, **Empleados**, **Ubicaciones**, **Actividades**, **Boletos** y **Tiendas**, además de varias tablas de relación que gestionan las interacciones entre estas entidades. Cada entidad en el diagrama está representada por un rectángulo que enumera sus atributos principales; por ejemplo, la entidad **Eventos** puede incluir atributos como *id*, *nombre*, *fecha* y *estado*.

Las relaciones entre las entidades se visualizan mediante líneas que conectan estos rectángulos, y los rombos describen el tipo de relación, como las relaciones uno a muchos o muchos a muchos. Por ejemplo, la relación entre **Eventos** y **Empleados** podría gestionarse a través de una tabla intermedia como **Asignaciones de Empleados**, que vincula empleados con eventos específicos en los que trabajan.

Las claves primarias y foráneas están claramente indicadas dentro de cada entidad, subrayando cómo se asegura la integridad referencial a través del sistema. Esto incluye relaciones directas como la de **Visitantes** que compran **Boletos**, así como relaciones más complejas como la que vincula **Productos** y **Tiendas** a través de la tabla **Inventarios**.

Este diagrama no solo nos ayuda a identificar rápidamente las relaciones y dependencias entre las diferentes partes de nuestra base de datos, sino que también es crucial para el diseño eficiente del sistema, asegurando que todas las interacciones y transacciones se manejen de manera coherente.

**Grafico:**


generalcategories
id INT
category_name VARCHAR(10...
type ENUM(...)
Indexes
products
id INT
shop_id INT
name VARCHAR(100)
description TEXT
manufacturer VARCHAR(10...
type VARCHAR(50)
available_quantity INT
price DECIMAL(10,2)
Indexes
inventoryite...
id INT
name VARCHAR(100)
quantity INT
status_id INT
event_id INT
Indexes
discounts
id INT
shop_id INT
product_id INT
discount_type ENUM(...)
discount_value DECIMAL(10,...
start_date DATETIME
end_date DATETIME
Indexes
ticketbooths
id INT
event_id INT
location VARCHAR(255)
contact_number VARCHAR(2...
manager_id INT
Indexes
activityparticipa...
id INT
activity_id INT
visitor_id INT
is_winner TINYINT(1)
Indexes
cosplayscor...
id INT
participant_id INT
judge_id INT
score DECIMAL(4,2)
Indexes
tickets
id INT
name VARCHAR(100)
price DECIMAL(10,2)
age_classification_id INT
additional_cost DECIMAL(10,...
status_id INT
ticket_booth_id INT
customer_id INT
Indexes
locations
id INT
country VARCHAR(100)
city VARCHAR(100)
address VARCHAR(25...
Indexes
cosplaycategori...
id INT
name VARCHAR(100)
description TEXT
Indexes
events
id INT
name VARCHAR(255)
date_time DATETIME
organizer_id INT
age_classification_id I...
status ENUM(...)
Indexes
orders
id INT
visitor_id INT
shop_id INT
cash_register_id INT
cashier_id INT
total_value DECIMAL(10,...
status ENUM(...)
Indexes
employeeeeventassignme...
id INT
employee_id INT
event_id INT
Indexes
ageclassificatio...
id INT
classification VARCHAR(10)
Indexes
employees
id INT
name VARCHAR(255)
identification VARCHAR(50)
birth_date DATE
role_id INT
status_id INT
Indexes
eventactivities
id INT
name VARCHAR(100)
type ENUM(...)
number_of_participants I...
event_id INT
start_time DATETIME
manager_id INT
category_id INT
Indexes
roles
id INT
name VARCHAR(100)
Indexes
employeestatushisto...
id INT
employee_id INT
status_id INT
change_date DATETIME
Indexes
inventoryitemstatus...
id INT
status_name VARCHAR(50)
Indexes
judges
id INT
name VARCHAR(255)
event_id INT
Indexes
organizers
id INT
name VARCHAR(255)
contact_info TEXT
Indexes
eventlocatio...
event_id INT
location_id INT
Indexes
shops
id INT
name VARCHAR(100)
category ENUM(...)
event_id INT
manager_id INT
Indexes
employeestatus...
id INT
status_name VARCHAR(50)
Indexes
cashregisters
id INT
status ENUM(...)
operator_id INT
opening_amount DECIMAL(10,...
closing_amount DECIMAL(10,2)
Indexes
dishtypes
id INT
type_name VARCHAR(10...
Indexes
roleactivityassignme...
role_id INT
activity_id INT
assigned_date DATETIME
Indexes
dishingredie...
dish_id INT
ingredient_id INT
quantity_required INT
Indexes
ticketstatus...
id INT
status ENUM(...)
Indexes
orderdetails
id INT
order_id INT
product_id INT
quantity INT
unit_price DECIMAL(10,...
Indexes
prizes
id INT
type VARCHAR(100)
description TEXT
value DECIMAL(10,...
status ENUM(...)
activity_id INT
participant_id INT
Indexes
triviaquestions
id INT
question TEXT
correct_answer TEXT
category VARCHAR(10...
difficulty ENUM(...)
event_id INT
category_id INT
Indexes
visitors
id INT
name VARCHAR(255)
identification_document VARCHAR(5...
gender ENUM(...)
birth_date DATE
email VARCHAR(100)
phone_number VARCHAR(20)
Indexes
ingredients
id INT
name VARCHAR(100)
available_quantity I...
Indexes
activities
id INT
name VARCHAR(100)
Indexes
triviaroun...
id INT
event_id INT
round_number INT
participant1_id INT
participant2_id INT
winner_id INT
Indexes
eventcapacities
id INT
event_id INT
max_capacity_people INT
max_capacity_shops INT
max_capacity_restaurants I...
Indexes
dishes
id INT
restaurant_id INT
name VARCHAR(100)
price DECIMAL(10,2)
type_id INT
preparation_time I...
Indexes

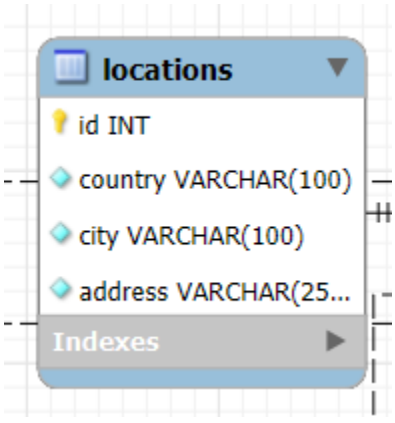
***Tablas***

**Locations**:

La tabla "Locations" está diseñada para almacenar información detallada sobre los diferentes lugares donde se llevan a cabo los eventos de anime. Esta tabla es fundamental para gestionar la logística y asegurar que todos los eventos se realicen en ubicaciones adecuadas y bien organizadas.

La estructura de la tabla "Locations" incluye los siguientes campos:

- **id**: Un identificador único para cada ubicación, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando que cada registro en la tabla sea único.
- **country**: Un campo de tipo varchar(100) que almacena el nombre del país donde se encuentra la ubicación. Este dato es crucial para categorizar las ubicaciones por región geográfica.
- **city**: Un campo de tipo varchar(100) que registra la ciudad de la ubicación. Este dato es esencial para la logística y la planificación del evento.
- **address**: Un campo de tipo varchar(255) que contiene la dirección específica de la ubicación. Este dato es vital para que los asistentes y organizadores puedan encontrar el lugar del evento con facilidad.

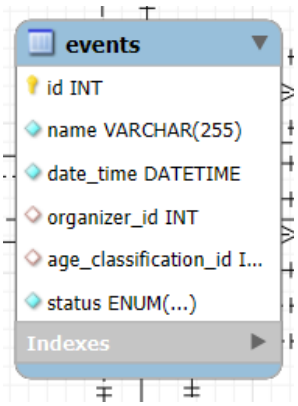


**Events**:

La tabla "Events" se utiliza para almacenar la información básica relacionada con cada uno de los eventos de anime que se organizan. Es crucial para coordinar todos los aspectos del evento y asegurarse de que cada uno se ejecute según lo planeado.

La estructura de la tabla "Events" incluye los siguientes campos:

- **id**: Un identificador único para cada evento, que se incrementa automáticamente. Este campo es la clave primaria, garantizando la unicidad de cada evento en el sistema.
- **name**: Un campo de tipo varchar(255) que almacena el nombre del evento. Este dato es fundamental para la identificación y promoción del evento.
- **date_time**: Un campo de tipo datetime que registra la fecha y hora en que se llevará a cabo el evento. Es esencial para la planificación y programación de las actividades del evento.
- **organizer_id**: Un campo de tipo int que refiere al organizador del evento. Este campo es clave para asociar el evento con la entidad responsable de su realización.
- **age_classification_id**: Un campo de tipo int que refiere a la clasificación por edades del evento. Es importante para garantizar que el evento sea adecuado para la audiencia prevista.
- **status**: Un campo de tipo enum que indica el estado actual del evento (por ejemplo, ACTIVE, FINISHED, PENDING). Este dato es importante para la gestión y el seguimiento del evento.

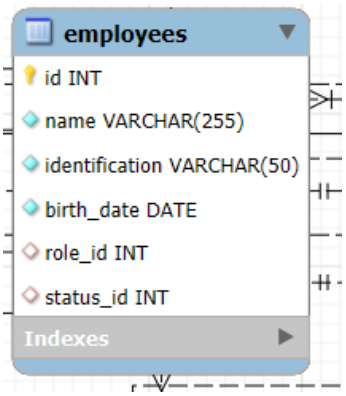


**Employees**:

La tabla "Employees" está diseñada para almacenar la información del personal que trabaja en los eventos de anime. Es fundamental para la gestión de recursos humanos y para asegurar que cada evento tenga el personal adecuado.

La estructura de la tabla "Employees" incluye los siguientes campos:

- **id**: Un identificador único para cada empleado, que se incrementa automáticamente. Este campo actúa como la clave primaria, garantizando la unicidad de cada registro.
- **name**: Un campo de tipo varchar(255) que almacena el nombre del empleado. Es esencial para la identificación y gestión del personal.
- **identification**: Un campo de tipo varchar(50) que registra el documento de identificación del empleado. Este dato es crucial para la verificación de identidad y seguridad.
- **birth_date**: Un campo de tipo date que almacena la fecha de nacimiento del empleado. Es importante para determinar la edad y las condiciones laborales del personal.
- **role_id**: Un campo de tipo int que refiere al rol del empleado dentro del evento. Este dato es vital para la asignación de tareas y responsabilidades.
- **status_id**: Un campo de tipo int que indica el estado laboral del empleado. Este campo es clave para gestionar el historial y la situación actual del personal.

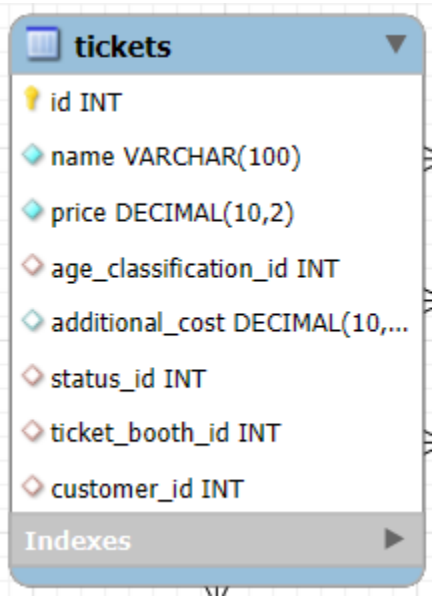


**Tickets**:

La tabla "Tickets" se utiliza para almacenar la información relacionada con los boletos de entrada a los eventos de anime. Es crucial para la gestión de ventas y el control de acceso a los eventos.

La estructura de la tabla "Tickets" incluye los siguientes campos:

- **id**: Un identificador único para cada boleto, que se incrementa automáticamente. Este campo sirve como la clave primaria, asegurando la unicidad de cada boleto en el sistema.
- **name**: Un campo de tipo varchar(100) que almacena el nombre o tipo de boleto. Este dato es esencial para categorizar y gestionar los diferentes tipos de entradas disponibles.
- **price**: Un campo de tipo decimal que registra el precio del boleto. Es crucial para la contabilidad y la gestión financiera del evento.
- **age_classification_id**: Un campo de tipo int que refiere a la clasificación por edades aplicable al boleto. Es importante para asegurar que las entradas sean vendidas a las personas adecuadas.
- **additional_cost**: Un campo de tipo decimal que puede registrar un costo adicional asociado al boleto. Este dato es relevante para entradas con características especiales o servicios extra.
- **status_id**: Un campo de tipo int que indica el estado del boleto (por ejemplo, PAID, RESERVED). Es vital para el control del inventario de boletos y la administración de reservas.
- **ticket_booth_id**: Un campo de tipo int que refiere a la taquilla donde se vendió el boleto. Este dato es importante para la gestión de puntos de venta.
- **customer_id**: Un campo de tipo int que refiere al visitante que adquirió el boleto. Este campo es esencial para asociar cada boleto con el cliente correspondiente.



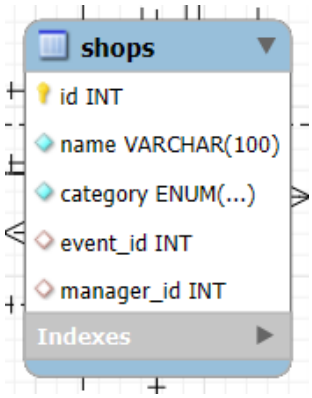

**Shops**:

La tabla "Shops" está destinada a almacenar información sobre las tiendas y restaurantes que participan en los eventos de anime. Es fundamental para gestionar las relaciones con los comercios y organizar los espacios de venta y servicios.

La estructura de la tabla "Shops" incluye los siguientes campos:

- **id**: Un identificador único para cada comercio, que se incrementa automáticamente. Este campo es la clave primaria y asegura que cada comercio registrado sea único.
- **name**: Un campo de tipo varchar(100) que almacena el nombre del comercio. Este dato es esencial para la identificación y promoción de los comercios participantes.
- **category**: Un campo de tipo enum que indica si el comercio es una tienda o un restaurante. Es importante para organizar los comercios y distribuir los espacios de manera adecuada.
- **event_id**: Un campo de tipo int que refiere al evento en el que participa el comercio. Este dato es crucial para asociar cada tienda o restaurante con el evento correspondiente.

- **manager_id**: Un campo de tipo int que refiere al empleado que gestiona el comercio. Este dato es importante para la administración y supervisión del comercio durante el evento.

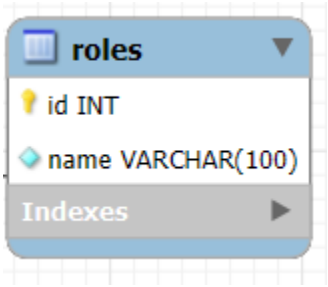



**Roles**:

La tabla "Roles" está diseñada para almacenar información sobre los diferentes roles que pueden asumir los empleados en los eventos de anime. Esta tabla es fundamental para asignar y gestionar las responsabilidades del personal, asegurando que cada tarea sea desempeñada por la persona adecuada.

La estructura de la tabla "Roles" incluye los siguientes campos:

- **id**: Un identificador único para cada rol, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando que cada registro en la tabla sea único.
- **name**: Un campo de tipo varchar(100) que almacena el nombre del rol. Este dato es esencial para identificar y asignar las funciones correspondientes a los empleados durante los eventos.

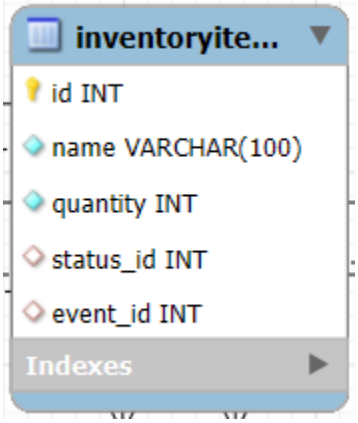


**InventoryItems**:

La tabla "InventoryItems" está destinada a almacenar información sobre los artículos del inventario que se utilizan durante los eventos de anime. Esta tabla es crucial para la gestión de recursos, asegurando que todos los materiales y equipos necesarios estén disponibles y en buen estado.

La estructura de la tabla "InventoryItems" incluye los siguientes campos:

- **id**: Un identificador único para cada artículo de inventario, que se incrementa automáticamente. Este campo sirve como la clave primaria, garantizando la unicidad de cada registro.
- **name**: Un campo de tipo varchar(100) que almacena el nombre del artículo. Es fundamental para identificar y catalogar cada elemento del inventario.
- **quantity**: Un campo de tipo int que registra la cantidad disponible de cada artículo. Este dato es crucial para el control del stock y para asegurar que no haya escasez de recursos durante los eventos.
- **status_id**: Un campo de tipo int que refiere al estado del artículo en el inventario. Es importante para gestionar la condición de los artículos, como si están en buen estado, en reparación o fuera de servicio.
- **event_id**: Un campo de tipo int que refiere al evento en el que se utiliza el artículo. Este dato es esencial para asociar cada artículo con su uso específico durante un evento.

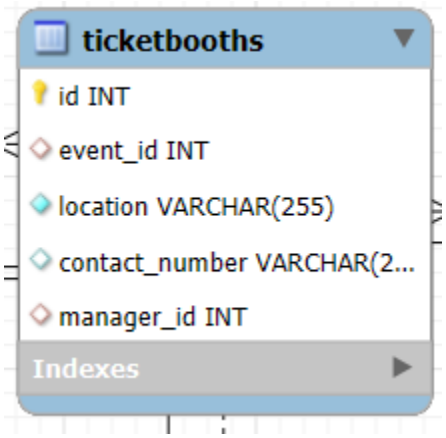


**TicketsBooths**:

La tabla "TicketBooths" se utiliza para almacenar información sobre las taquillas donde se venden los boletos para los eventos de anime. Esta tabla es fundamental para la organización y gestión de los puntos de venta de entradas, asegurando un control eficiente de las ventas y accesos.

La estructura de la tabla "TicketBooths" incluye los siguientes campos:

- **id**: Un identificador único para cada taquilla, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro en la tabla.
- **event_id**: Un campo de tipo int que refiere al evento para el cual se venden boletos en la taquilla. Este dato es esencial para asociar la taquilla con el evento correspondiente.
- **location**: Un campo de tipo varchar(255) que almacena la ubicación específica de la taquilla dentro del evento. Es importante para guiar a los visitantes y organizar eficientemente los puntos de venta.
- **contact_number**: Un campo de tipo varchar(20) que registra el número de contacto de la taquilla. Este dato es útil para la comunicación y coordinación durante el evento.
- **manager_id**: Un campo de tipo int que refiere al empleado responsable de la taquilla. Este dato es clave para la gestión y supervisión de las operaciones de venta.

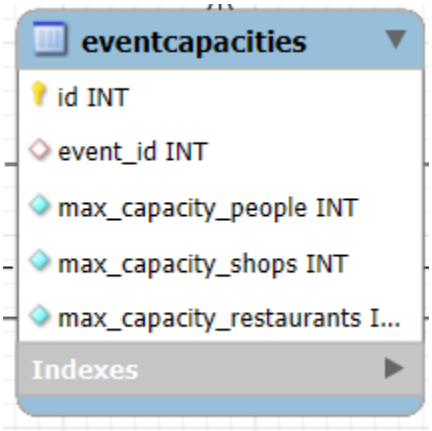


**EventCapacities**:

La tabla "EventCapacities" está diseñada para almacenar información sobre la capacidad máxima de personas, tiendas y restaurantes que pueden participar o estar presentes en un evento de anime. Esta tabla es crucial para asegurar que los eventos no excedan su capacidad y que se mantenga un control adecuado del aforo.

La estructura de la tabla "EventCapacities" incluye los siguientes campos:

- **id**: Un identificador único para cada capacidad registrada, que se incrementa automáticamente. Este campo sirve como la clave primaria, garantizando la unicidad de cada registro.
- **event_id**: Un campo de tipo int que refiere al evento al que se le asigna la capacidad. Este dato es esencial para asociar la capacidad con el evento correspondiente.
- **max_capacity_people**: Un campo de tipo int que registra la capacidad máxima de personas permitidas en el evento. Es crucial para garantizar la seguridad y cumplimiento de las normativas de aforo.
- **max_capacity_shops**: Un campo de tipo int que indica la cantidad máxima de tiendas permitidas en el evento. Este dato es importante para organizar el espacio comercial de manera eficiente.
- **max_capacity_restaurants**: Un campo de tipo int que registra la cantidad máxima de restaurantes permitidos en el evento. Es vital para la planificación y distribución del área de alimentos y bebidas.

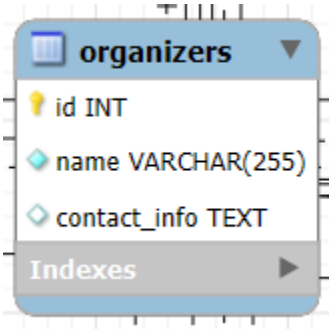


**Organizers**:

La tabla "Organizers" almacena información sobre las entidades o personas responsables de la organización de los eventos de anime. Esta tabla es fundamental para gestionar y coordinar todos los aspectos del evento, desde la planificación hasta la ejecución.

La estructura de la tabla "Organizers" incluye los siguientes campos:

- **id**: Un identificador único para cada organizador, que se incrementa automáticamente. Este campo actúa como la clave primaria, garantizando que cada registro sea único.
- **name**: Un campo de tipo varchar(255) que almacena el nombre del organizador. Este dato es esencial para la identificación y referencia de la entidad o persona a cargo del evento.
- **contact_info**: Un campo de tipo text que registra la información de contacto del organizador. Este dato es crucial para la comunicación y coordinación durante todas las fases del evento.

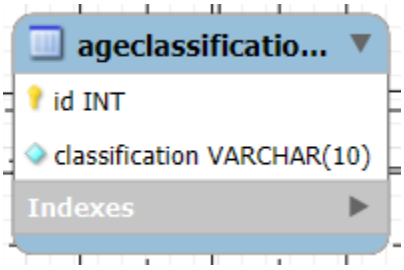



**AgeClassifications**:

La tabla "AgeClassifications" está diseñada para almacenar las clasificaciones por edades que se aplican a los eventos de anime. Esta tabla es fundamental para garantizar que los eventos sean adecuados para las diferentes audiencias, cumpliendo con las normativas y expectativas de los asistentes.

La estructura de la tabla "AgeClassifications" incluye los siguientes campos:

- **id**: Un identificador único para cada clasificación por edad, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando que cada registro en la tabla sea único.
- **classification**: Un campo de tipo varchar(10) que almacena la clasificación por edad (por ejemplo, "PG", "18+"). Este dato es esencial para categorizar los eventos y controlar la entrada según la edad de los participantes.

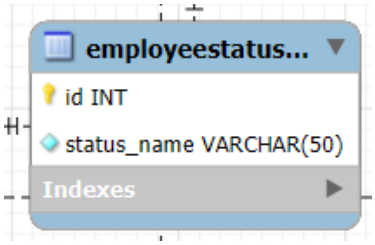



**EmployeeStatuses**:

La tabla "EmployeeStatuses" está destinada a almacenar los diferentes estados laborales de los empleados que trabajan en los eventos de anime. Esta tabla es crucial para la gestión de recursos humanos, permitiendo un seguimiento detallado de la situación laboral de cada empleado.

La estructura de la tabla "EmployeeStatuses" incluye los siguientes campos:

- **id**: Un identificador único para cada estado de empleado, que se incrementa automáticamente. Este campo sirve como la clave primaria, garantizando la unicidad de cada registro.
- **status_name**: Un campo de tipo varchar(50) que almacena el nombre del estado laboral (por ejemplo, "Activo", "En licencia"). Este dato es importante para gestionar y documentar la situación actual de los empleados



**InventoryItemStatuses**:

La tabla "InventoryItemStatuses" se utiliza para almacenar los diferentes estados de los artículos del inventario en los eventos de anime. Esta tabla es esencial para el control y seguimiento del estado de los recursos, asegurando que los artículos estén disponibles y en condiciones óptimas.

La estructura de la tabla "InventoryItemStatuses" incluye los siguientes campos:

- **id**: Un identificador único para cada estado de inventario, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro en la tabla.
- **status_name**: Un campo de tipo varchar(50) que almacena el nombre del estado del artículo (por ejemplo, "Disponible", "En reparación"). Este dato es vital para la gestión eficiente del inventario durante los eventos.

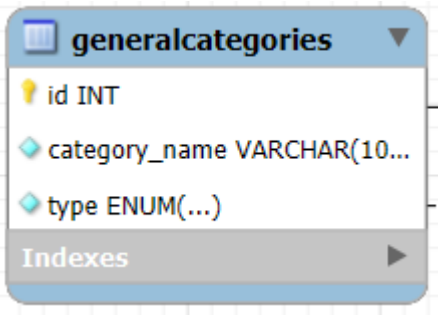

**GeneralCategories**:

La tabla "GeneralCategories" está diseñada para almacenar las categorías generales que se aplican a los diferentes elementos de los eventos de anime, como eventos, platillos, tiendas y actividades. Esta tabla es fundamental para la organización y clasificación de los diversos componentes del evento.

La estructura de la tabla "GeneralCategories" incluye los siguientes campos:

- **id**: Un identificador único para cada categoría, que se incrementa automáticamente. Este campo sirve como la clave primaria, garantizando la unicidad de cada registro.
- **category_name**: Un campo de tipo varchar(100) que almacena el nombre de la categoría. Este dato es esencial para clasificar y organizar los diferentes elementos del evento.

- **type**: Un campo de tipo enum que indica el tipo de elemento al que se aplica la categoría (por ejemplo, "EVENT", "DISH", "SHOP", "ACTIVITY"). Este dato es crucial para la correcta categorización y gestión de los recursos del evento.

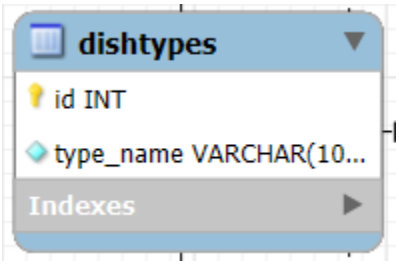



**DishTypes**:

La tabla "DishTypes" se utiliza para almacenar los diferentes tipos de platillos que se pueden ofrecer en los restaurantes durante los eventos de anime. Esta tabla es esencial para la gestión y clasificación del menú, asegurando una oferta variada y adecuada para los asistentes.

La estructura de la tabla "DishTypes" incluye los siguientes campos:

- **id**: Un identificador único para cada tipo de platillo, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **type_name**: Un campo de tipo varchar(100) que almacena el nombre del tipo de platillo (por ejemplo, "Entrante", "Principal", "Postre"). Este dato es fundamental para organizar y categorizar los elementos del menú.

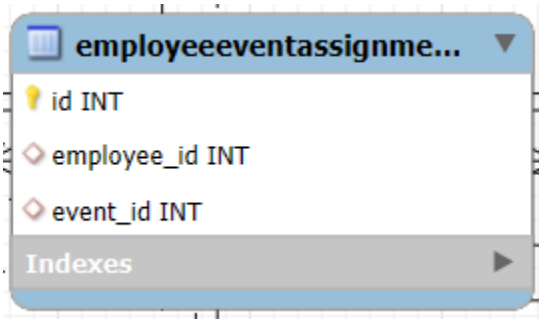



**EmployeeEventAssignments**:

La tabla "EmployeeEventAssignments" está destinada a almacenar las asignaciones de los empleados a los diferentes eventos de anime. Esta tabla es crucial para la gestión de recursos humanos, asegurando que cada evento cuente con el personal adecuado.

La estructura de la tabla "EmployeeEventAssignments" incluye los siguientes campos:

- **id**: Un identificador único para cada asignación, que se incrementa automáticamente. Este campo sirve como la clave primaria, garantizando la unicidad de cada registro.
- **employee_id**: Un campo de tipo int que refiere al empleado asignado al evento. Este dato es esencial para gestionar las asignaciones y responsabilidades del personal.
- **event_id**: Un campo de tipo int que refiere al evento al que está asignado el empleado. Este dato es importante para asociar las tareas y funciones con el evento correspondiente.




**EmployeeStatusHistory**:
La tabla "EmployeeStatusHistory" se utiliza para almacenar el historial de cambios de estado laboral de los empleados que trabajan en los eventos de anime. Esta tabla es esencial para documentar y seguir la trayectoria laboral de cada empleado.

La estructura de la tabla "EmployeeStatusHistory" incluye los siguientes campos:

- **id**: Un identificador único para cada registro de cambio de estado, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **employee_id**: Un campo de tipo int que refiere al empleado cuyo estado ha cambiado. Este dato es crucial para documentar la historia laboral del personal.
- **status_id**: Un campo de tipo int que indica el nuevo estado del empleado. Este dato es importante para registrar la situación laboral actualizada del personal.

- **change_date**: Un campo de tipo datetime que registra la fecha y hora en que se produjo el cambio de estado. Este dato es esencial para llevar un seguimiento cronológico de los cambios en la situación laboral.


**ActivityParticipants**:

La tabla "ActivityParticipants" está diseñada para almacenar información sobre los participantes en las actividades organizadas durante los eventos de anime. Esta tabla es crucial para gestionar y documentar la participación en concursos, trivias y otras actividades interactivas.

La estructura de la tabla "ActivityParticipants" incluye los siguientes campos:

- **id**: Un identificador único para cada participante, que se incrementa automáticamente. Este campo actúa como la clave primaria, garantizando la unicidad de cada registro.
- **activity_id**: Un campo de tipo int que refiere a la actividad en la que participa el visitante. Este dato es esencial para asociar a los participantes con las actividades correspondientes.
- **visitor_id**: Un campo de tipo int que refiere al visitante que participa en la actividad. Este dato es fundamental para documentar y gestionar la participación de los asistentes.
- **is_winner**: Un campo de tipo boolean que indica si el participante ha ganado la actividad. Este dato es importante para la adjudicación de premios y la organización de los resultados.

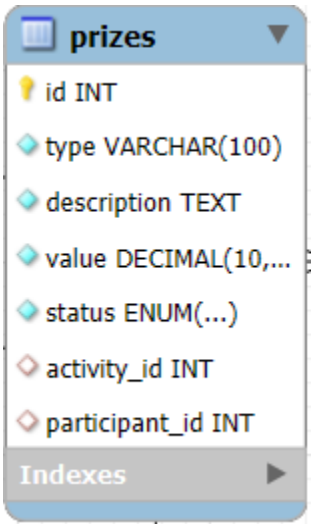

**Prizes**:

La tabla "Prizes" se utiliza para almacenar información sobre los premios que se otorgan en las actividades organizadas durante los eventos de anime. Esta tabla es esencial para la gestión y distribución de premios, asegurando que se entreguen a los ganadores de manera ordenada y documentada.

La estructura de la tabla "Prizes" incluye los siguientes campos:

- **id**: Un identificador único para cada premio, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **type**: Un campo de tipo varchar(100) que almacena el tipo de premio (por ejemplo, "Trofeo", "Medalla", "Dinero"). Este dato es crucial para categorizar y gestionar los premios.
- **description**: Un campo de tipo text que proporciona una descripción detallada del premio. Este dato es importante para documentar las características y valor del premio.
- **value**: Un campo de tipo decimal que registra el valor monetario del premio. Este dato es esencial para la contabilidad y gestión financiera del evento.
- **status**: Un campo de tipo enum que indica el estado del premio (por ejemplo, "AVAILABLE", "DELIVERED"). Este dato es vital para asegurar que los premios se entreguen correctamente a los ganadores.

- **activity_id**: Un campo de tipo int que refiere a la actividad en la que se otorga el premio. Este dato es fundamental para asociar cada premio con la actividad correspondiente.
- **participant_id**: Un campo de tipo int que refiere al participante que recibe el premio. Este dato es importante para documentar la adjudicación del premio.



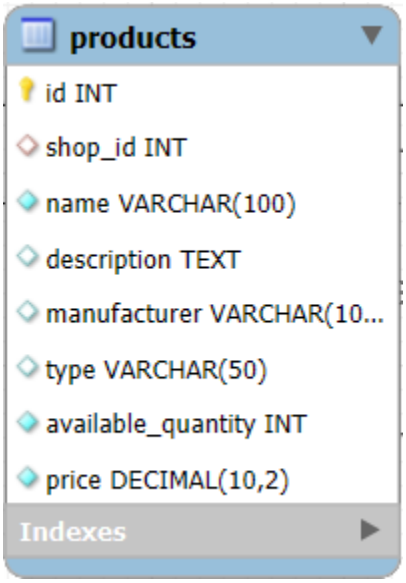

**Products**:

La tabla "Products" está destinada a almacenar información sobre los productos que se venden en las tiendas durante los eventos de anime. Esta tabla es crucial para la gestión del inventario y las ventas, asegurando que los asistentes tengan acceso a una variedad de artículos.

La estructura de la tabla "Products" incluye los siguientes campos:

- **id**: Un identificador único para cada producto, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **shop_id**: Un campo de tipo int que refiere a la tienda que vende el producto. Este dato es esencial para asociar cada producto con su punto de venta.
- **name**: Un campo de tipo varchar(100) que almacena el nombre del producto. Este dato es crucial para la identificación y promoción de los artículos.

- **description**: Un campo de tipo text que proporciona una descripción detallada del producto. Este dato es importante para informar a los compradores sobre las características del artículo.
- **manufacturer**: Un campo de tipo varchar(100) que registra el nombre del fabricante del producto. Este dato es relevante para la trazabilidad y calidad del producto.
- **type**: Un campo de tipo varchar(50) que indica el tipo de producto (por ejemplo, "Figura", "Ropa"). Este dato es fundamental para la categorización de los productos.
- **available_quantity**: Un campo de tipo int que registra la cantidad disponible del producto. Este dato es esencial para el control del stock y la gestión del inventario.
- **price**: Un campo de tipo decimal que registra el precio del producto. Este dato es crucial para la contabilidad y gestión de las ventas.

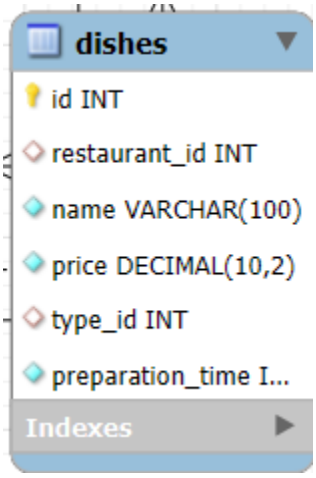



**Dishes**:

La tabla "Dishes" está diseñada para almacenar información sobre los platillos que se ofrecen en los restaurantes durante los eventos de anime. Esta tabla es fundamental para la gestión del menú y la oferta culinaria, asegurando que los asistentes tengan acceso a una variedad de opciones gastronómicas.

La estructura de la tabla "Dishes" incluye los siguientes campos:

- **id**: Un identificador único para cada platillo, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **restaurant_id**: Un campo de tipo int que refiere al restaurante que ofrece el platillo. Este dato es esencial para asociar cada platillo con su lugar de preparación.
- **name**: Un campo de tipo varchar(100) que almacena el nombre del platillo. Este dato es crucial para la identificación y promoción de los platillos en el menú.
- **price**: Un campo de tipo decimal que registra el precio del platillo. Este dato es esencial para la contabilidad y gestión de ventas en el restaurante.
- **type_id**: Un campo de tipo int que refiere al tipo de platillo. Este dato es importante para la categorización de los elementos del menú.
- **preparation_time**: Un campo de tipo int que indica el tiempo de preparación en minutos. Este dato es vital para la planificación de la cocina y la experiencia del cliente.

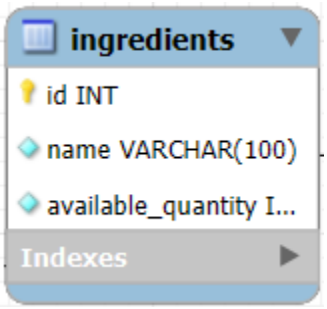



**Ingredients**:

La tabla "Ingredients" está destinada a almacenar información sobre los ingredientes que se utilizan en la preparación de platillos durante los eventos de anime. Esta tabla es esencial para la gestión de la cocina y el control de inventario, asegurando que todos los ingredientes estén disponibles cuando se necesiten.

La estructura de la tabla "Ingredients" incluye los siguientes campos:

- **id**: Un identificador único para cada ingrediente, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **name**: Un campo de tipo varchar(100) que almacena el nombre del ingrediente. Este dato es crucial para la identificación y gestión del inventario en la cocina.
- **available_quantity**: Un campo de tipo int que registra la cantidad disponible del ingrediente. Este dato es esencial para asegurar que la cocina tenga suficientes suministros para preparar los platillos.




**DishIngredients**:

La tabla "DishIngredients" se utiliza para almacenar la relación entre los platillos y los ingredientes necesarios para su preparación durante los eventos de anime. Esta tabla es crucial para la gestión de recetas y la planificación del menú en los restaurantes.

La estructura de la tabla "DishIngredients" incluye los siguientes campos:

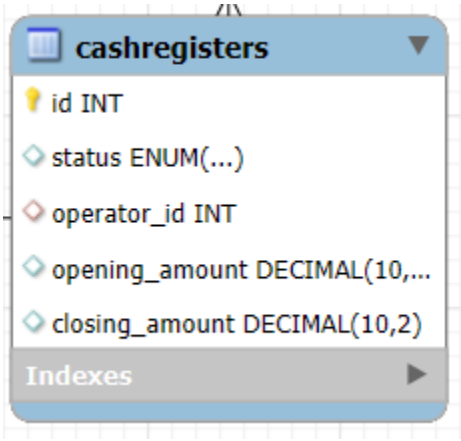
- **dish_id**: Un campo de tipo int que refiere al platillo que requiere el ingrediente. Este dato es esencial para asociar los ingredientes con los platillos correspondientes.
- **ingredient_id**: Un campo de tipo int que refiere al ingrediente necesario para el platillo. Este dato es importante para asegurar que se utilicen los ingredientes correctos en cada receta.
- **quantity_required**: Un campo de tipo int que indica la cantidad del ingrediente necesario para preparar el platillo. Este dato es vital para la planificación y control de la cocina.


**CashRegisters**:

La tabla "CashRegisters" está diseñada para almacenar información sobre las cajas registradoras utilizadas en los eventos de anime. Esta tabla es fundamental para la gestión financiera, asegurando que todas las transacciones se registren de manera adecuada y que las operaciones sean seguras.

La estructura de la tabla "CashRegisters" incluye los siguientes campos:

- **id**: Un identificador único para cada caja registradora, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **status**: Un campo de tipo enum que indica el estado de la caja registradora (por ejemplo, "ACTIVE", "INACTIVE"). Este dato es crucial para el control de la operación de las cajas registradoras durante el evento.
- **operator_id**: Un campo de tipo int que refiere al empleado que opera la caja registradora. Este dato es esencial para la gestión del personal y la seguridad financiera.
- **opening_amount**: Un campo de tipo decimal que registra el monto inicial en la caja registradora al comenzar la operación. Este dato es importante para la contabilidad y el control de efectivo.
- **closing_amount**: Un campo de tipo decimal que registra el monto final en la caja registradora al finalizar la operación. Este dato es vital para cerrar correctamente las operaciones y verificar los ingresos.



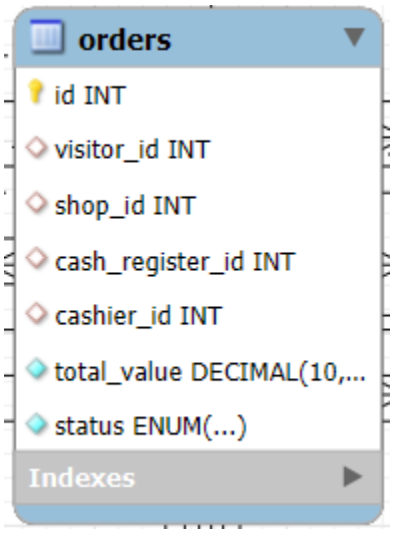

**Orders**:

La tabla "Orders" se utiliza para almacenar información sobre los pedidos realizados por los visitantes en las tiendas y restaurantes durante los eventos de anime. Esta tabla es fundamental para la gestión de ventas y la satisfacción del cliente, asegurando que todos los pedidos se procesen de manera eficiente.

La estructura de la tabla "Orders" incluye los siguientes campos:

- **id**: Un identificador único para cada pedido, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **visitor_id**: Un campo de tipo int que refiere al visitante que realiza el pedido. Este dato es esencial para asociar cada pedido con el cliente correspondiente.
- **shop_id**: Un campo de tipo int que refiere a la tienda o restaurante donde se realiza el pedido. Este dato es crucial para la gestión de ventas y el control de inventario.
- **cash_register_id**: Un campo de tipo int que refiere a la caja registradora que procesa el pedido. Este dato es importante para la gestión financiera y la seguridad de las transacciones.
- **cashier_id**: Un campo de tipo int que refiere al empleado que procesa el pedido en la caja registradora. Este dato es esencial para la gestión del personal y la satisfacción del cliente.
- **total_value**: Un campo de tipo decimal que registra el valor total del pedido. Este dato es vital para la contabilidad y el control de ingresos.

- **status**: Un campo de tipo enum que indica el estado del pedido (por ejemplo, "REGISTERED", "PAID", "DELIVERED"). Este dato es crucial para asegurar que los pedidos se gestionen de manera efectiva y que los clientes reciban su compra.

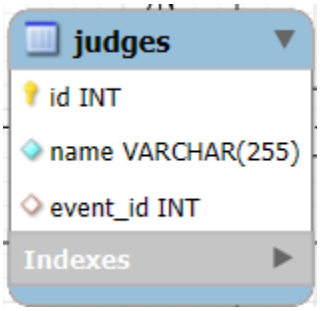



**Judges**:

La tabla "Judges" está destinada a almacenar información sobre los jueces que participan en la evaluación de actividades durante los eventos de anime. Esta tabla es esencial para la organización de competencias como cosplay y trivias, asegurando que los resultados sean justos y bien documentados.

La estructura de la tabla "Judges" incluye los siguientes campos:

- **id**: Un identificador único para cada juez, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **name**: Un campo de tipo varchar(255) que almacena el nombre del juez. Este dato es esencial para la identificación y asignación de responsabilidades durante las competencias.
- **event_id**: Un campo de tipo int que refiere al evento en el que el juez participa. Este dato es importante para asociar al juez con las actividades correspondientes del evento.

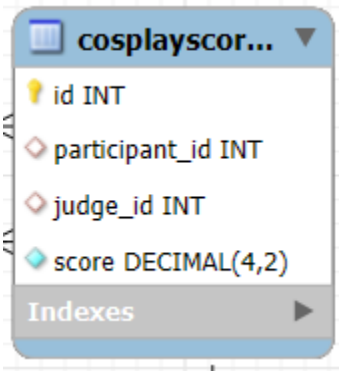



**CosplayScores**:

La tabla "CosplayScores" se utiliza para almacenar las puntuaciones otorgadas por los jueces a los participantes en concursos de cosplay durante los eventos de anime. Esta tabla es fundamental para la gestión de competencias, asegurando que los resultados se registren de manera precisa y transparente.

La estructura de la tabla "CosplayScores" incluye los siguientes campos:

- **id**: Un identificador único para cada registro de puntuación, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **participant_id**: Un campo de tipo int que refiere al participante que recibe la puntuación. Este dato es esencial para documentar el rendimiento de los concursantes.
- **judge_id**: Un campo de tipo int que refiere al juez que otorga la puntuación. Este dato es importante para garantizar la transparencia y trazabilidad de las evaluaciones.
- **score**: Un campo de tipo decimal que registra la puntuación otorgada al participante. Este dato es crucial para determinar los ganadores de la competencia y asegurar que las evaluaciones sean justas.

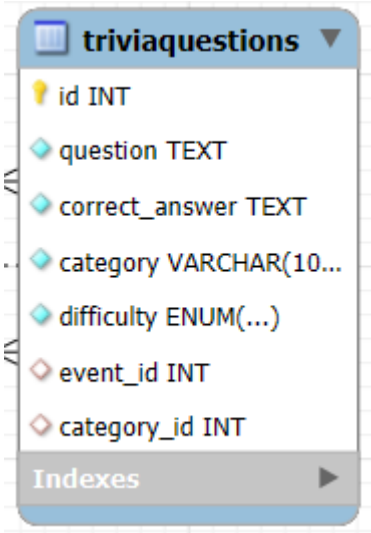


**TriviaQuestions**:

La tabla "TriviaQuestions" está destinada a almacenar las preguntas utilizadas en las competencias de trivia durante los eventos de anime. Esta tabla es esencial para la organización y gestión de las actividades de trivia, asegurando que las preguntas estén bien categorizadas y documentadas.

La estructura de la tabla "TriviaQuestions" incluye los siguientes campos:

- **id**: Un identificador único para cada pregunta de trivia, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **question**: Un campo de tipo text que almacena el texto de la pregunta. Este dato es esencial para la presentación y evaluación de las preguntas durante la competencia.
- **correct_answer**: Un campo de tipo text que registra la respuesta correcta a la pregunta. Este dato es crucial para la evaluación de las respuestas de los participantes.
- **category**: Un campo de tipo varchar(100) que indica la categoría a la que pertenece la pregunta (por ejemplo, "Anime", "Manga"). Este dato es importante para la organización de la trivia y la selección de preguntas.
- **difficulty**: Un campo de tipo enum que indica el nivel de dificultad de la pregunta (por ejemplo, "EASY", "INTERMEDIATE", "HARD"). Este dato es vital para equilibrar la competencia y asegurar que las preguntas se adapten al nivel de los participantes.
- **event_id**: Un campo de tipo int que refiere al evento en el que se utiliza la pregunta. Este dato es esencial para asociar las preguntas con la actividad correspondiente.
- **category_id**: Un campo de tipo int que refiere a la categoría general de la pregunta. Este dato es importante para la correcta categorización y gestión de las preguntas de trivia.



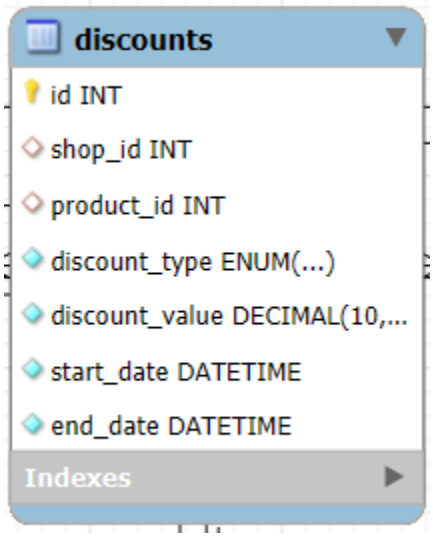

**Discounts**:

La tabla "Discounts" se utiliza para almacenar información sobre los descuentos ofrecidos en las tiendas y restaurantes durante los eventos de anime. Esta tabla es fundamental para la gestión de promociones y ventas, asegurando que los descuentos se apliquen correctamente y que los clientes puedan aprovechar las ofertas.

La estructura de la tabla "Discounts" incluye los siguientes campos:

- **id**: Un identificador único para cada descuento, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **shop_id**: Un campo de tipo int que refiere a la tienda o restaurante que ofrece el descuento. Este dato es esencial para asociar el descuento con el punto de venta correspondiente.
- **product_id**: Un campo de tipo int que refiere al producto al que se aplica el descuento. Este dato es crucial para garantizar que el descuento se aplique de manera correcta y transparente.
- **discount_type**: Un campo de tipo enum que indica el tipo de descuento (por ejemplo, "PERCENTAGE", "FIXED_AMOUNT", "BUNDLE"). Este dato es importante para la correcta aplicación y gestión del descuento.

- **discount_value**: Un campo de tipo decimal que registra el valor del descuento. Este dato es esencial para calcular el precio final del producto y asegurar que los clientes reciban el descuento correspondiente.
- **start_date**: Un campo de tipo datetime que indica la fecha y hora de inicio del descuento. Este dato es vital para la planificación y gestión de las promociones.
- **end_date**: Un campo de tipo datetime que indica la fecha y hora de finalización del descuento. Este dato es importante para asegurar que el descuento se aplique solo dentro del período promocional especificado.



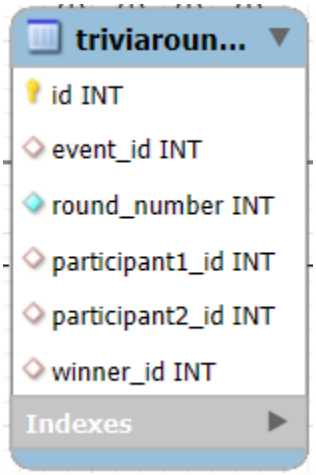

**TriviaRounds**:

La tabla "TriviaRounds" está destinada a almacenar información sobre las rondas de competencia en las actividades de trivia durante los eventos de anime. Esta tabla es esencial para la organización y gestión de las competencias, asegurando que las rondas se desarrollen de manera ordenada y justa.

La estructura de la tabla "TriviaRounds" incluye los siguientes campos:

- **id**: Un identificador único para cada ronda de trivia, que se incrementa automáticamente. Este campo actúa como la clave primaria, asegurando la unicidad de cada registro.
- **event_id**: Un campo de tipo int que refiere al evento en el que se lleva a cabo la ronda de trivia. Este dato es esencial para asociar la ronda con la actividad correspondiente.

- **round_number**: Un campo de tipo int que indica el número de la ronda dentro de la competencia. Este dato es importante para la organización y seguimiento de las diferentes etapas de la trivia.
- **participant1_id**: Un campo de tipo int que refiere al primer participante en la ronda. Este dato es crucial para la gestión de la competencia y la adjudicación de puntos.
- **participant2_id**: Un campo de tipo int que refiere al segundo participante en la ronda. Este dato es esencial para documentar y gestionar la competencia de manera adecuada.
- **winner_id**: Un campo de tipo int que refiere al participante que ganó la ronda. Este dato es vital para determinar los ganadores de la competencia y asegurar que los resultados se registren de manera precisa.



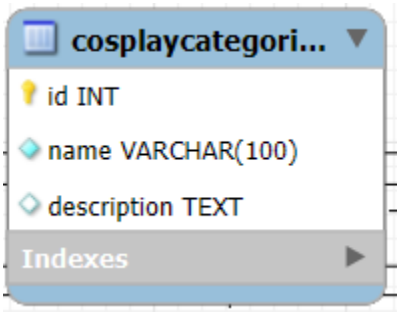

**CosplayCategories**:

La tabla "CosplayCategories" está diseñada para almacenar información sobre las diferentes categorías de concursos de cosplay que se realizan durante los eventos de anime. Esta tabla es fundamental para la organización y clasificación de las competencias, asegurando que los participantes se inscriban en la categoría adecuada.

La estructura de la tabla "CosplayCategories" incluye los siguientes campos:

- **id**: Un identificador único para cada categoría de cosplay, que se incrementa automáticamente. Este campo actúa como la clave primaria, garantizando la unicidad de cada registro.

- **name**: Un campo de tipo varchar(100) que almacena el nombre de la categoría de cosplay. Este dato es esencial para identificar y clasificar las diferentes competencias de cosplay.
- **description**: Un campo de tipo text que proporciona una descripción detallada de la categoría. Este dato es importante para informar a los participantes sobre los requisitos y criterios de cada categoría.



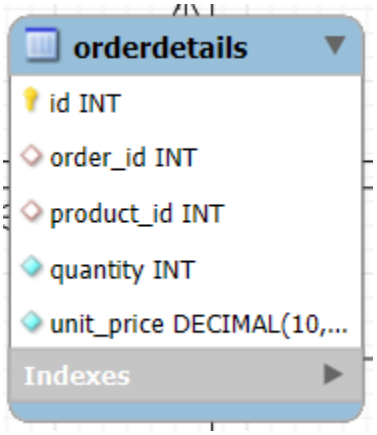

**OrderDetails**:

La tabla "OrderDetails" se utiliza para almacenar información detallada sobre los productos incluidos en cada pedido realizado en las tiendas y restaurantes durante los eventos de anime. Esta tabla es crucial para la gestión de inventarios y ventas, asegurando que cada pedido se registre con precisión.

La estructura de la tabla "OrderDetails" incluye los siguientes campos:

- **id**: Un identificador único para cada detalle de pedido, que se incrementa automáticamente. Este campo actúa como la clave primaria, garantizando la unicidad de cada registro.
- **order_id**: Un campo de tipo int que refiere al pedido al que pertenece el detalle. Este dato es esencial para asociar cada producto con el pedido correspondiente.
- **product_id**: Un campo de tipo int que refiere al producto incluido en el pedido. Este dato es crucial para el control del inventario y la gestión de ventas.
- **quantity**: Un campo de tipo int que registra la cantidad de productos solicitados en el pedido. Este dato es importante para calcular el total del pedido y asegurar que se entreguen los productos correctos.

- **unit_price**: Un campo de tipo decimal que almacena el precio unitario del producto. Este dato es vital para calcular el costo total del pedido y para la contabilidad.

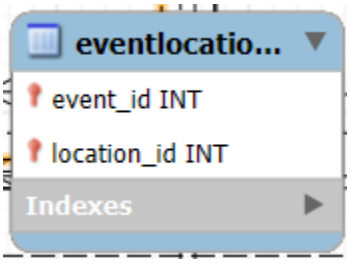



**EventLocations**:

La tabla "EventLocations" está destinada a almacenar la relación entre los eventos y las ubicaciones donde se llevan a cabo. Esta tabla es fundamental para la organización logística de los eventos, asegurando que cada evento esté asociado con su lugar correspondiente.

La estructura de la tabla "EventLocations" incluye los siguientes campos:

- **event_id**: Un campo de tipo int que refiere al evento. Este dato es esencial para asociar cada evento con su ubicación correspondiente.
- **location_id**: Un campo de tipo int que refiere a la ubicación donde se lleva a cabo el evento. Este dato es crucial para la planificación y gestión del espacio en los eventos.



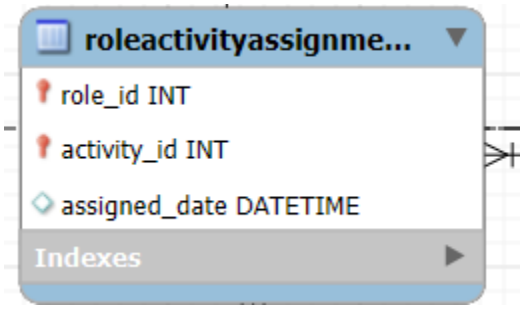

**RoleActivityAssignments**:

La tabla "RoleActivityAssignments" se utiliza para almacenar la asignación de actividades específicas a los diferentes roles desempeñados por los empleados durante los eventos de

anime. Esta tabla es esencial para la gestión de recursos humanos, asegurando que cada tarea sea realizada por la persona adecuada.

La estructura de la tabla "RoleActivityAssignments" incluye los siguientes campos:

- **role_id**: Un campo de tipo int que refiere al rol del empleado. Este dato es esencial para definir las responsabilidades y tareas asignadas a cada rol.
- **activity_id**: Un campo de tipo int que refiere a la actividad asignada al rol. Este dato es crucial para garantizar que las actividades se lleven a cabo de manera ordenada y eficiente.
- **assigned_date**: Un campo de tipo datetime que registra la fecha y hora en que se asignó la actividad al rol. Este dato es importante para documentar el flujo de trabajo y asegurar el cumplimiento de las tareas asignadas.

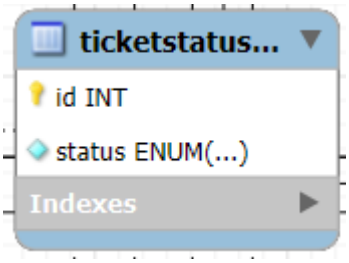



**TicketStatuses**:

La tabla "TicketStatuses" está diseñada para almacenar los diferentes estados en los que puede encontrarse un boleto durante los eventos de anime. Esta tabla es fundamental para la gestión de entradas, asegurando que se pueda seguir y controlar el estado de cada boleto.

La estructura de la tabla "TicketStatuses" incluye los siguientes campos:

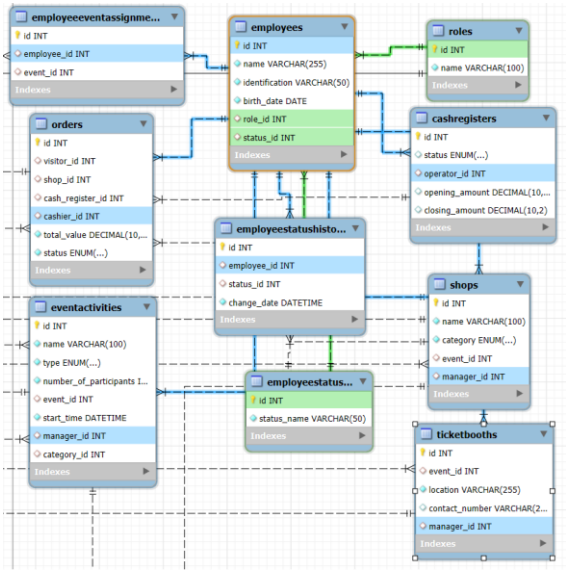
- **id**: Un identificador único para cada estado de boleto, que se incrementa automáticamente. Este campo actúa como la clave primaria, garantizando la unicidad de cada registro.
- **status**: Un campo de tipo enum que indica el estado del boleto (por ejemplo, "PAID", "RESERVED"). Este dato es esencial para el control de la emisión y venta de boletos, asegurando que se gestionen correctamente.




***Relaciones entre tablas***

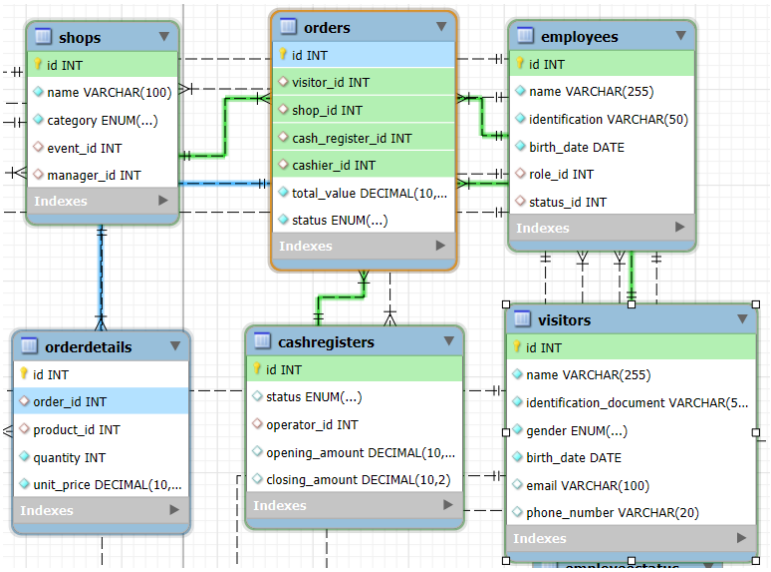
**Employees**

La tabla Employees juega un papel crucial en la gestión de personal dentro de los eventos de anime, estando altamente interconectada con múltiples tablas en la base de datos. A través de la clave foránea role_id, se relaciona con la tabla Roles, permitiendo asignar funciones específicas a cada empleado. De manera similar, la clave foránea status_id conecta Employees con EmployeeStatuses, facilitando el seguimiento de la situación laboral de cada miembro del equipo. Además, Employees se relaciona con EmployeeEventAssignments y EmployeeStatusHistory mediante la clave employee_id, lo que permite tanto la asignación de personal a eventos como el registro histórico de cambios en su estado laboral. La relación con TicketBooths y EventActivities, a través de la clave manager_id, asegura que cada taquilla y actividad tenga un responsable designado, mientras que la clave cashier_id en Orders y operator_id en CashRegisters permite identificar al personal encargado de procesar pedidos y manejar transacciones. Adicionalmente, la relación con Shops a través de manager_id otorga a los empleados la responsabilidad de gestionar tiendas o restaurantes en los eventos. Estas múltiples relaciones subrayan la importancia de la tabla Employees en la organización y operación eficiente de los eventos de anime.



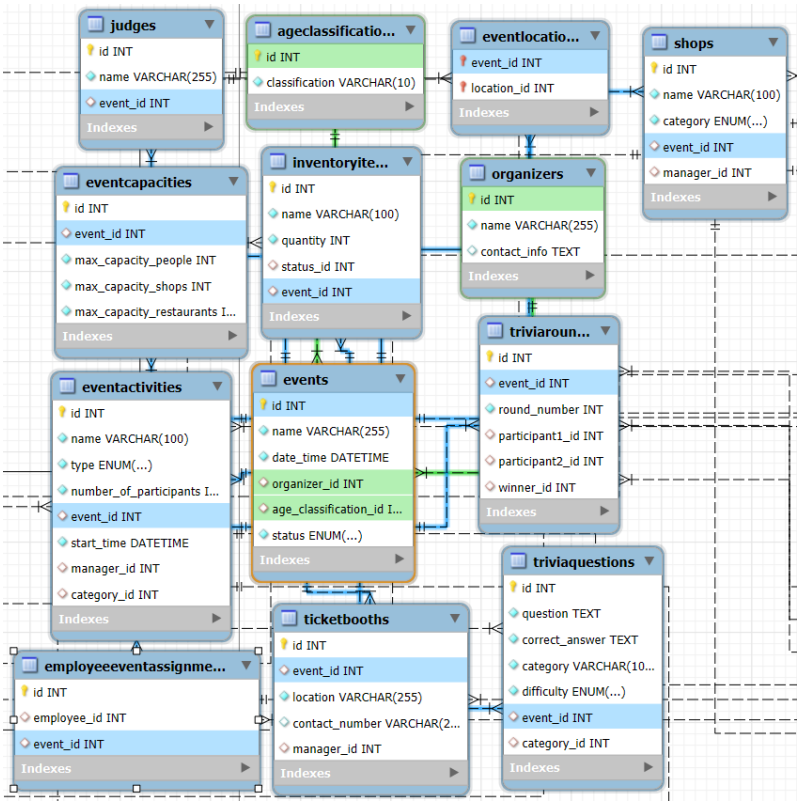

## Orders

La tabla Orders se encuentra en el centro de la gestión de transacciones durante los eventos de anime, vinculándose con varias tablas clave para asegurar un control preciso y detallado. A través de la relación con la tabla Visitors mediante la clave visitor_id, cada pedido se asocia directamente con el visitante que lo realizó, garantizando un registro claro de las transacciones. La conexión con la tabla Shops, mediante la clave shop_id, permite relacionar cada pedido con la tienda o restaurante correspondiente, facilitando la gestión de ventas y el control de inventario. Además, la relación con CashRegisters, a través de la clave cash_register_id, asegura que todas las transacciones estén vinculadas a la caja registradora correcta, lo que es esencial para la contabilidad. La clave cashier_id conecta Orders con la tabla Employees, permitiendo identificar al empleado que procesó el pedido, lo que es crucial para la gestión del personal y la trazabilidad de las operaciones. Finalmente, la relación con OrderDetails, mediante la clave order_id, permite desglosar cada pedido en sus componentes específicos, registrando detalladamente los productos, cantidades y precios unitarios involucrados en cada transacción.



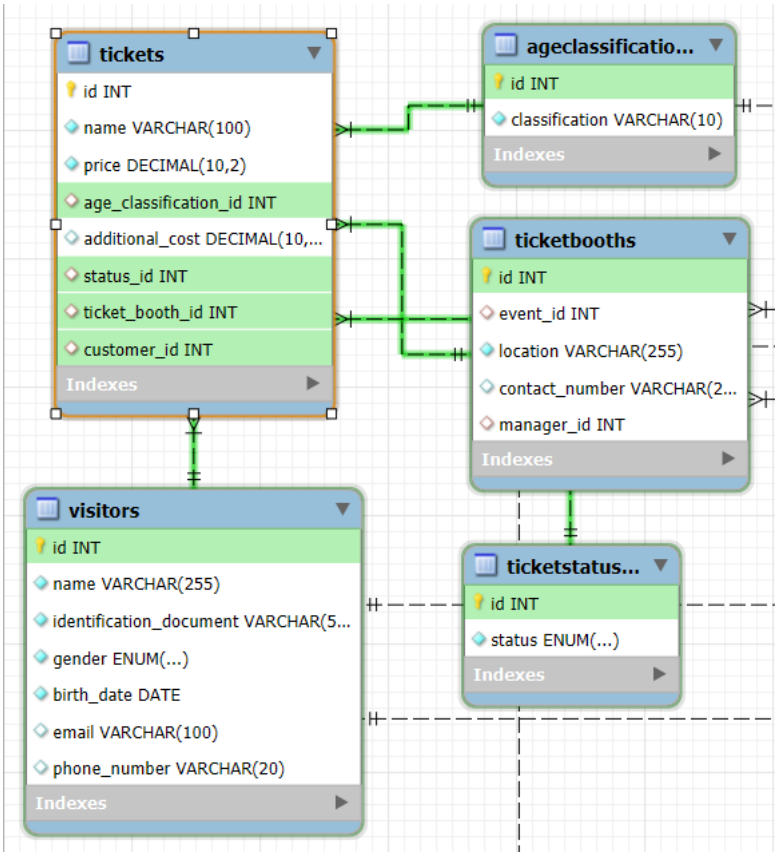

**Events:**

La tabla Events es una de las tablas centrales en la base de datos de gestión de eventos de anime, ya que está relacionada con múltiples tablas clave, lo que permite una organización y coordinación integral de los eventos. A través de la relación con la tabla Organizers, mediante la clave foránea organizer_id, cada evento se asocia con su organizador, lo que facilita la gestión y el control administrativo del evento. La relación con la tabla AgeClassifications, a través de la clave age_classification_id, permite clasificar los eventos según la edad adecuada para los participantes, garantizando el cumplimiento de las normativas y expectativas del público. Además, la tabla Events se relaciona con EventLocations mediante la clave event_id, lo que vincula cada evento con su ubicación específica, asegurando una planificación logística eficiente. La relación con EventCapacities, también a través de event_id, permite definir y controlar la capacidad máxima del evento en términos de asistentes, tiendas y restaurantes, garantizando que no se excedan los límites establecidos. Asimismo, la conexión con EmployeeEventAssignments a través de event_id permite asignar personal específico a cada evento, asegurando que haya suficiente apoyo operativo. Finalmente, la relación con EventActivities, mediante event_id, vincula las actividades planificadas con el evento correspondiente, facilitando una gestión ordenada y coherente de las diversas actividades que se llevarán a cabo durante el evento. Estas múltiples relaciones hacen de la tabla Events el núcleo organizativo de la base de datos, asegurando que cada aspecto del evento esté debidamente coordinado y documentado.



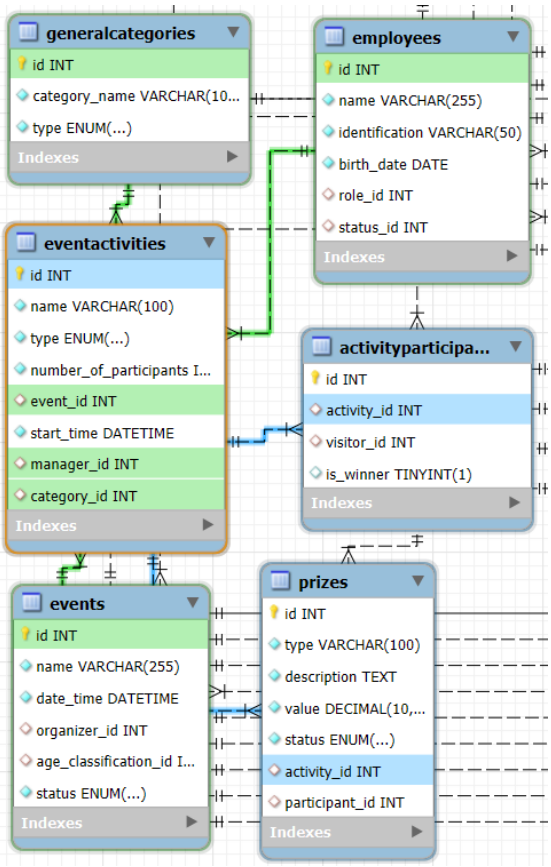

## Tickets

La tabla Tickets desempeña un papel crucial en la gestión de las entradas durante los eventos de anime, al estar interconectada con varias tablas clave que aseguran un control eficiente y detallado. A través de la relación con la tabla AgeClassifications, mediante la clave foránea age_classification_id, cada boleto se clasifica según la edad permitida para el evento, garantizando que se cumplan las restricciones de edad necesarias. La relación con la tabla TicketStatuses, mediante la clave status_id, permite gestionar y rastrear el estado de cada boleto, ya sea que esté pagado, reservado u otra condición relevante, lo que es esencial para el control de accesos. Además, la tabla Tickets está vinculada con la tabla TicketBooths a través de la clave ticket_booth_id, lo que facilita la identificación de la taquilla en la que se vendió cada boleto, mejorando la gestión de puntos de venta. Finalmente, la relación con la tabla Visitors, mediante la clave customer_id, asocia cada boleto con el visitante que lo adquirió, asegurando un registro claro y preciso de quién compró cada entrada, lo cual es fundamental para la trazabilidad y el servicio al cliente.



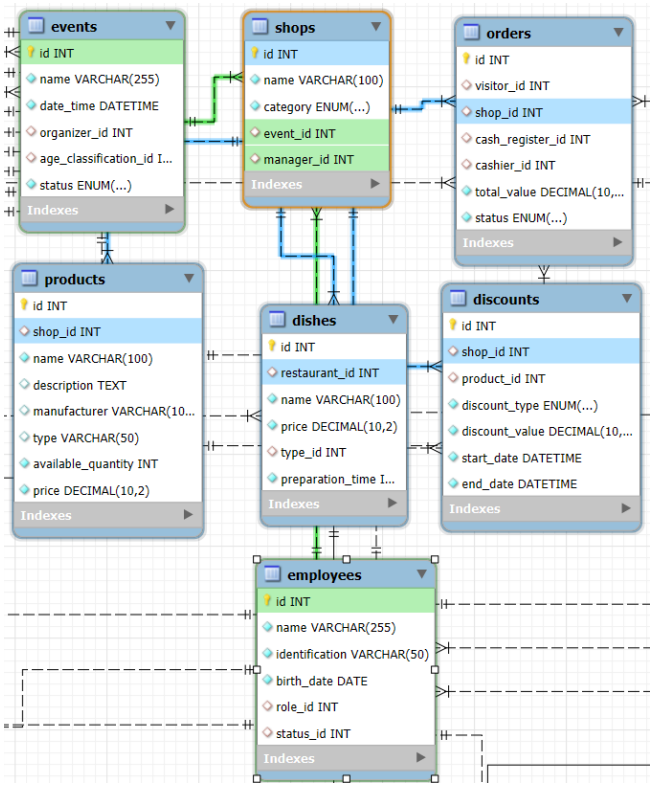

**EventActivity**

La tabla EventActivities es fundamental para la gestión de actividades dentro de los eventos de anime, al estar conectada con varias tablas clave que facilitan una organización eficiente. A través de la relación con la tabla Events, mediante la clave foránea event_id, cada actividad se asocia con el evento específico en el que se llevará a cabo, lo que permite una planificación coordinada y coherente de las actividades programadas. La relación con la tabla Employees, a través de la clave manager_id, permite asignar un empleado como responsable de la actividad, asegurando que haya una gestión adecuada y supervisión durante su ejecución. Además, la tabla EventActivities está vinculada con la tabla GeneralCategories mediante la clave category_id, lo que facilita la categorización de las actividades dentro de un esquema general, ayudando a organizar las actividades por tipo y temática. Estas relaciones hacen que la tabla EventActivities sea esencial para la planificación detallada y la ejecución de las actividades dentro de los eventos de anime, asegurando que cada actividad esté bien coordinada y gestionada.



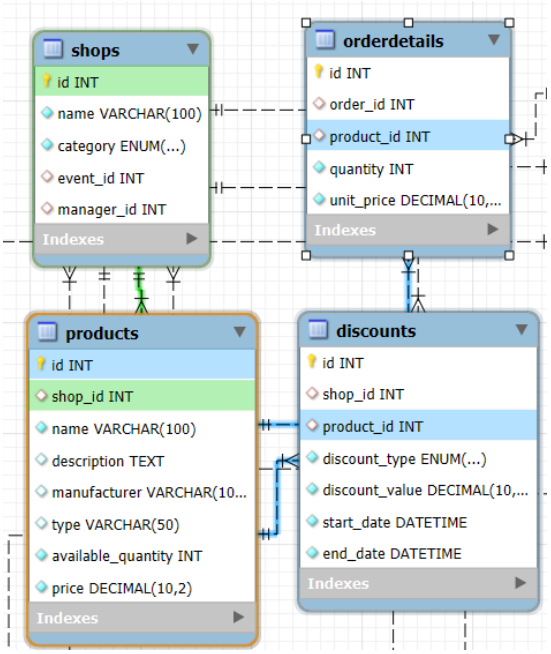

**Shops**

La tabla Shops es crucial para la gestión de los comercios, como tiendas y restaurantes, dentro de los eventos de anime, y está relacionada con otras tablas clave que aseguran su correcto funcionamiento. A través de la relación con la tabla Events, mediante la clave foránea event_id, cada comercio se asocia con un evento específico, permitiendo una organización efectiva de los espacios comerciales durante el evento. La relación con la tabla Employees, mediante la clave manager_id, asigna un gerente a cada comercio, garantizando que haya un responsable encargado de las operaciones diarias del comercio.




**Products**

la tabla Products se enfoca en la gestión de los productos que se venden en estos comercios y está vinculada a la tabla Shops a través de la clave foránea shop_id, lo que permite asociar cada producto con la tienda o restaurante que lo ofrece. Además, la tabla Products se relaciona con la tabla Discounts mediante la clave product_id, lo que permite aplicar y gestionar descuentos específicos en los productos, facilitando las promociones y el control de precios durante el evento. Estas relaciones aseguran que tanto la gestión de los comercios como la de los productos y descuentos sean manejadas de manera eficiente y coordinada dentro de los eventos de anime.




***Inserciones de datos***

Las inserciones realizadas en la base de datos de eventos de anime abarcan una serie detallada de operaciones SQL que cubren múltiples tablas interrelacionadas. Estas inserciones incluyen datos esenciales como ubicaciones, clasificaciones por edad, estados laborales y de inventario, categorías generales, roles de empleados, actividades, tipos de platillos, organizadores de eventos, y más. Además, se han insertado registros específicos para eventos, asignaciones de personal, cambios de estado laboral, elementos de inventario, taquillas, visitantes, estados de boletos, tickets, actividades dentro de los eventos, participantes, premios, tiendas, productos, platillos, ingredientes, cajas registradoras, pedidos, jueces, puntuaciones de cosplay, preguntas de trivia, descuentos, rondas de trivia, categorías de cosplay y detalles de pedidos. Cada inserción está diseñada para simular datos reales, con el objetivo de validar tanto el diseño como la funcionalidad del software, asegurando que todas las operaciones y flujos de datos dentro del sistema funcionen correctamente en un entorno de gestión de eventos complejos.

```sql
-- Insert into Locations
INSERT INTO Locations (country, city, address)
VALUES ('Japan', 'Tokyo', '123 Anime St, Akihabara');

-- Insert into AgeClassifications
INSERT INTO AgeClassifications (classification)
VALUES ('PG-13');

-- Insert into EmployeeStatuses
INSERT INTO EmployeeStatuses (status_name)
VALUES ('Active');

-- Insert into InventoryItemStatuses
INSERT INTO InventoryItemStatuses (status_name)
VALUES ('Available');

-- Insert into GeneralCategories
INSERT INTO GeneralCategories (category_name, type)
VALUES ('Action Figures', 'SHOP');

-- Insert into Roles
INSERT INTO Roles (name)
VALUES ('Event Manager');

-- Insert into Activities
INSERT INTO Activities (name)
VALUES ('Cosplay Contest');

-- Insert into DishTypes
INSERT INTO DishTypes (type_name)
VALUES ('Appetizer');

-- Insert into Organizers
INSERT INTO Organizers (name, contact_info)
VALUES ('AnimeCon', 'info@animecon.com');

-- Insert into Events
INSERT INTO Events (name, date_time, organizer_id, age_classification_id,
status)
VALUES ('Anime Expo 2024', '2024-08-30 10:00:00', 1, 1, 'ACTIVE');

-- Insert into EventLocations
INSERT INTO EventLocations (event_id, location_id)
VALUES (1, 1);
```

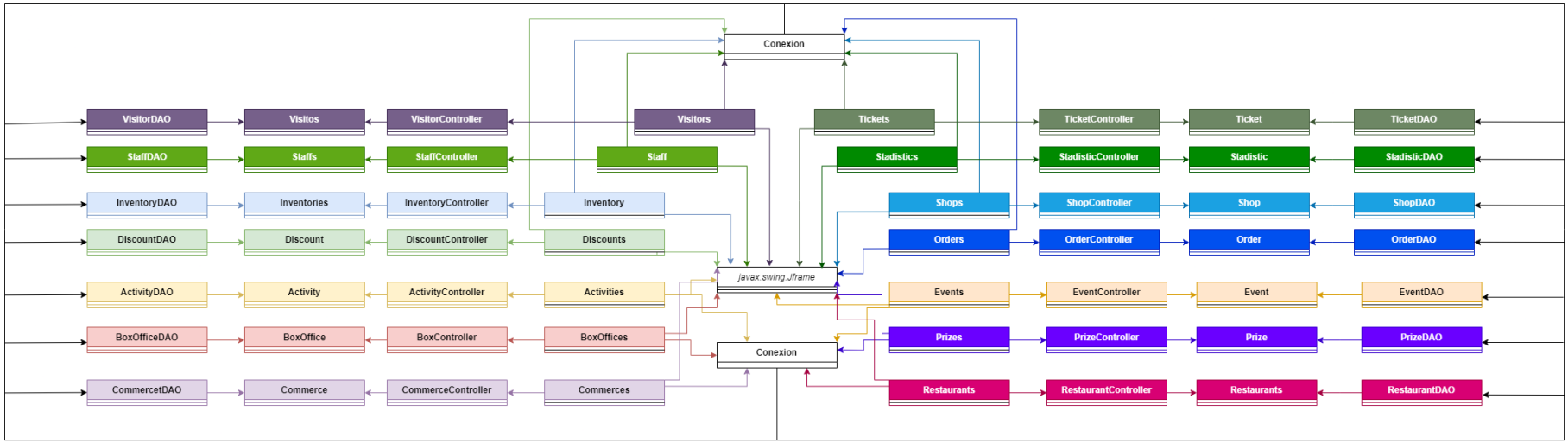
```sql
-- Insert into EventCapacities
INSERT INTO EventCapacities (event_id, max_capacity_people,
max_capacity_shops, max_capacity_restaurants)
VALUES (1, 10000, 50, 30);

-- Insert into RoleActivityAssignments
INSERT INTO RoleActivityAssignments (role_id, activity_id)
VALUES (1, 1);

-- Insert into Employees
INSERT INTO Employees (name, identification, birth_date, role_id, status_id)
VALUES ('John Doe', '123456789', '1990-05-20', 1, 1);

-- Insert into EmployeeEventAssignments
INSERT INTO EmployeeEventAssignments (employee_id, event_id)
VALUES (1, 1);

-- Insert into EmployeeStatusHistory
INSERT INTO EmployeeStatusHistory (employee_id, status_id, change_date)
VALUES (1, 1, '2024-08-01 09:00:00');

-- Insert into InventoryItems
INSERT INTO InventoryItems (name, quantity, status_id, event_id)
VALUES ('Projector', 5, 1, 1);

-- Insert into TicketBooths
INSERT INTO TicketBooths (event_id, location, contact_number, manager_id)
VALUES (1, 'Main Entrance', '123-456-7890', 1);

-- Insert into Visitors
INSERT INTO Visitors (name, identification_document, gender, birth_date,
email, phone_number)
VALUES ('Jane Smith', '987654321', 'FEMALE', '1995-07-15',
'jane.smith@example.com', '098-765-4321');

-- Insert into TicketStatuses
INSERT INTO TicketStatuses (status)
VALUES ('PAID');

-- Insert into Tickets
INSERT INTO Tickets (name, price, age_classification_id, additional_cost,
status_id, ticket_booth_id, customer_id)
VALUES ('VIP Pass', 150.00, 1, 10.00, 1, 1, 1);

-- Insert into EventActivities
```

```sql
INSERT INTO EventActivities (name, type, number_of_participants, event_id,
start_time, manager_id, category_id)
VALUES ('Cosplay Competition', 'COSPLAY', 50, 1, '2024-08-30 14:00:00', 1,
1);

-- Insert into ActivityParticipants
INSERT INTO ActivityParticipants (activity_id, visitor_id, is_winner)
VALUES (1, 1, FALSE);

-- Insert into Prizes
INSERT INTO Prizes (type, description, value, status, activity_id,
participant_id)
VALUES ('Trophy', 'First Place Cosplay Trophy', 100.00, 'AVAILABLE', 1, 1);

-- Insert into Shops
INSERT INTO Shops (name, category, event_id, manager_id)
VALUES ('Otaku Shop', 'SHOP', 1, 1);

-- Insert into Products
INSERT INTO Products (shop_id, name, description, manufacturer, type,
available_quantity, price)
VALUES (1, 'Naruto Action Figure', 'High quality Naruto figure', 'Anime Toys
Co.', 'Figure', 100, 25.00);

-- Insert into Dishes
INSERT INTO Dishes (restaurant_id, name, price, type_id, preparation_time)
VALUES (1, 'Ramen', 12.00, 1, 15);

-- Insert into Ingredients
INSERT INTO Ingredients (name, available_quantity)
VALUES ('Noodles', 100);

-- Insert into DishIngredients
INSERT INTO DishIngredients (dish_id, ingredient_id, quantity_required)
VALUES (1, 1, 2);

-- Insert into CashRegisters
INSERT INTO CashRegisters (status, operator_id, opening_amount,
closing_amount)
VALUES ('ACTIVE', 1, 100.00, NULL);

-- Insert into Orders
INSERT INTO Orders (visitor_id, shop_id, cash_register_id, cashier_id,
total_value, status)
VALUES (1, 1, 1, 1, 50.00, 'PAID');
```

```sql
-- Insert into Judges
INSERT INTO Judges (name, event_id)
VALUES ('Sasuke Uchiha', 1);

-- Insert into CosplayScores
INSERT INTO CosplayScores (participant_id, judge_id, score)
VALUES (1, 1, 9.5);

-- Insert into TriviaQuestions
INSERT INTO TriviaQuestions (question, correct_answer, category, difficulty,
event_id, category_id)
VALUES ('What year did Naruto debut?', '2002', 'Anime', 'EASY', 1, 1);

-- Insert into Discounts
INSERT INTO Discounts (shop_id, product_id, discount_type, discount_value,
start_date, end_date)
VALUES (1, 1, 'PERCENTAGE', 10.00, '2024-08-25 00:00:00', '2024-08-30
23:59:59');

-- Insert into TriviaRounds
INSERT INTO TriviaRounds (event_id, round_number, participant1_id,
participant2_id, winner_id)
VALUES (1, 1, 1, 1, 1);

-- Insert into CosplayCategories
INSERT INTO CosplayCategories (name, description)
VALUES ('Best Character Representation', 'Award for the best representation
of a character.');

-- Insert into OrderDetails
INSERT INTO OrderDetails (order_id, product_id, quantity, unit_price)
VALUES (1, 1, 2, 25.00);
```

# Java

## Diagrama de clases



**Jframe**

El diagrama de clases que se describe está estructurado en torno a la clase principal javax.swing.JFrame, que es una clase estándar en Java utilizada para crear ventanas en aplicaciones gráficas. Esta clase sirve como base para diversas subclases que representan distintos módulos de la aplicación, permitiendo gestionar diferentes funcionalidades dentro de una interfaz gráfica de usuario.

**Clase Principal: javax.swing.JFrame**

javax.swing.JFrame es la clase padre en este diagrama, y su función principal es proporcionar las herramientas necesarias para construir una ventana dentro de una aplicación Java. Esta clase facilita la disposición de componentes gráficos, la gestión de eventos de usuario, y la interacción con el sistema operativo, constituyendo así la base de la interfaz gráfica.

**Métodos Heredados**

Las subclases derivadas de JFrame heredan una serie de métodos que son fundamentales para la configuración y funcionamiento de la interfaz:

1. **initComponents()**:
    - Este método es responsable de inicializar y configurar los componentes de la interfaz gráfica, como botones, etiquetas y cuadros de texto. Es esencial para definir cómo se verá y funcionará la ventana al ser creada.
2. **setLocationRelativeTo(null)**:
    - Este método se utiliza para centrar la ventana en la pantalla del usuario. Al pasar null como parámetro, la ventana se posiciona automáticamente en el centro.
3. **eventNameTextActionPerformed(java.awt.event.ActionEvent evt)**:

- Este método gestiona la acción realizada cuando el usuario interactúa con el campo de texto destinado a ingresar el nombre del evento, ya sea al escribir o al presionar Enter.

4. **AgeTextActionPerformed(java.awt.event.ActionEvent evt)**:
    - Este método se activa cuando el usuario interactúa con el campo de texto donde se ingresa la edad, manejando cualquier acción relacionada con este input.
5. **organizerTextActionPerformed(java.awt.event.ActionEvent evt)**:
    - Este método se encarga de las acciones relacionadas con el campo de texto en el que se ingresa el nombre del organizador del evento.
6. **dateTextActionPerformed(java.awt.event.ActionEvent evt)**:
    - Este método maneja la interacción del usuario con el campo de texto relacionado con la fecha del evento.
7. **back1ActionPerformed(java.awt.event.ActionEvent evt)**:
    - Este método gestiona la acción del botón "Back", que normalmente se utiliza para regresar a la pantalla anterior o al menú principal.
8. **statusTextActionPerformed(java.awt.event.ActionEvent evt)**:
    - Este método se utiliza para manejar las acciones relacionadas con el campo de texto que muestra o ingresa el estado del evento u otros elementos dentro de la aplicación.
9. **editTextActionPerformed(java.awt.event.ActionEvent evt)**:
    - Este método se encarga de las acciones que se realizan cuando el usuario interactúa con el campo de texto destinado a editar información existente.

**Subclases**

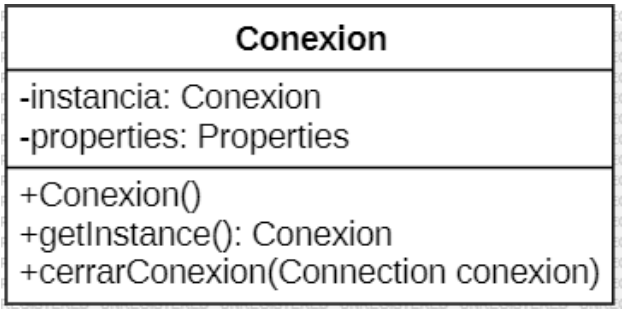
Las subclases que heredan de javax.swing.JFrame y que utilizan los métodos mencionados están diseñadas para gestionar diferentes partes de la aplicación. Cada una de estas subclases

se enfoca en un área específica de la aplicación, proporcionando la interfaz gráfica necesaria para interactuar con el usuario:

1. **Welcome**: Maneja la pantalla de bienvenida de la aplicación.
2. **Tickets**: Administra la interfaz relacionada con la gestión y visualización de boletos.
3. **Events**: Controla la pantalla destinada a la gestión de eventos en la aplicación.
4. **Activities**: Gestiona la interfaz para la administración de actividades dentro de los eventos.
5. **Management**: Se encarga de la interfaz para la gestión administrativa general del sistema.
6. **Instructions**: Proporciona una pantalla con instrucciones o guías para los usuarios.
7. **Shops**: Administra la interfaz relacionada con las tiendas y los productos durante el evento.
8. **Software**: Podría estar enfocada en la gestión de aspectos técnicos o de software del evento.
9. **BoxOffice**: Maneja la interfaz de la taquilla, donde se realizan las ventas de boletos.
10. **Inventory**: Administra la pantalla para la gestión del inventario de productos y recursos.
11. **Staff**: Gestiona la interfaz para la administración del personal que participa en los eventos.

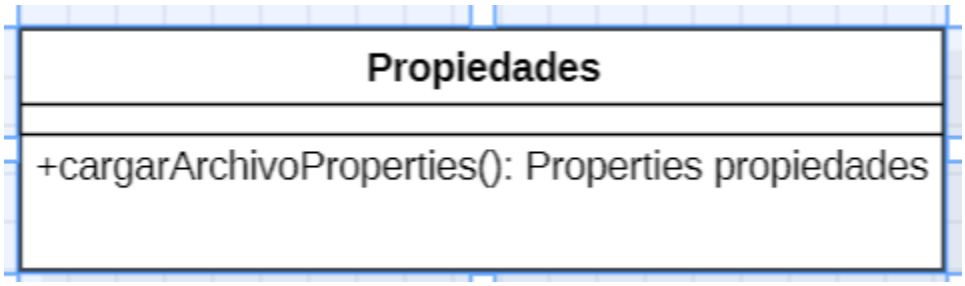
**Conexión a la base de datos**

La clase Conexion en este código es una implementación de un patrón Singleton que gestiona la conexión a una base de datos utilizando propiedades configuradas en un archivo externo. Esta clase garantiza que solo exista una instancia de la conexión a la base de datos durante la ejecución del programa, lo cual ayuda a optimizar recursos y a centralizar la gestión de la conexión. Incluye métodos para conectar a la base de datos y cerrar la conexión de manera segura, manejando posibles errores de manera eficiente.



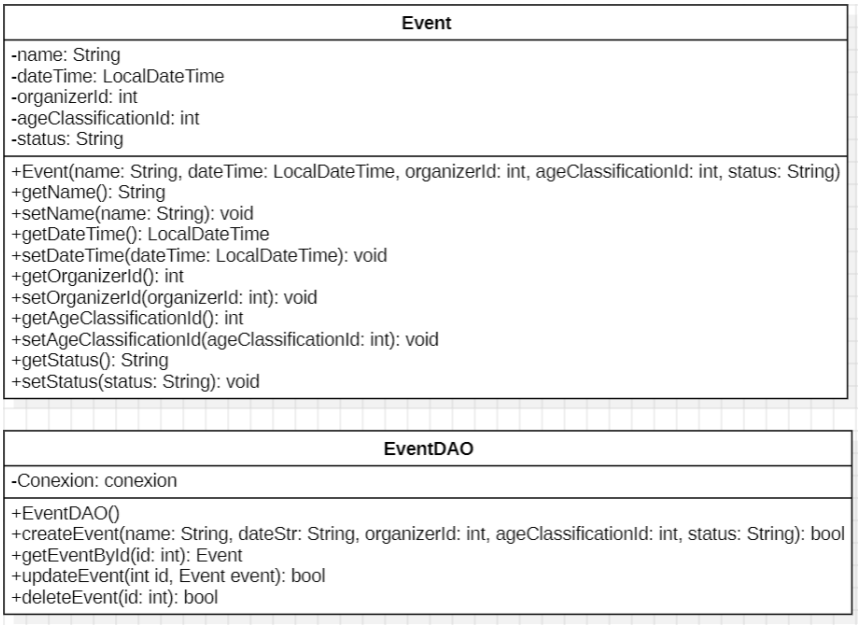

**Properties**

La clase Propiedades se encarga de cargar y gestionar los archivos de configuración que contienen las propiedades necesarias para establecer la conexión a la base de datos. Utiliza un objeto de tipo Properties para leer el archivo ConexionDB.properties desde el classpath, verificando si el archivo existe y lanzando una excepción en caso contrario. Este diseño permite centralizar la configuración de la conexión, facilitando su manejo y modificación sin necesidad de cambiar el código fuente.



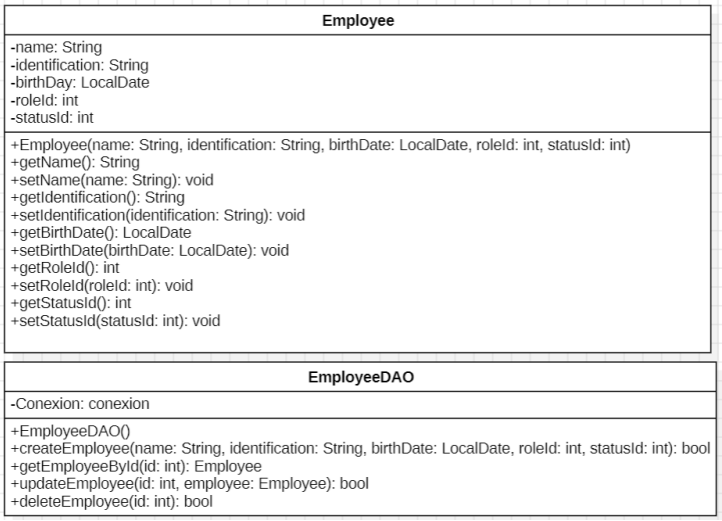

**Clases Event y EventDAO**

La clase Event define un modelo para los eventos, encapsulando atributos como el nombre, la fecha y hora, el organizador, la clasificación por edades y el estado del evento. Esta clase permite la creación y manipulación de instancias de eventos, facilitando el acceso y modificación de sus atributos mediante métodos getter y setter. Por otro lado, la clase EventDAO se encarga de la interacción con la base de datos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los eventos. Utiliza la clase Conexion para establecer la conexión a la base de datos, y mediante el uso de sentencias SQL preparadas, maneja las operaciones necesarias para persistir los datos de los eventos en la base de datos, gestionando adecuadamente las posibles excepciones que puedan ocurrir durante el proceso.



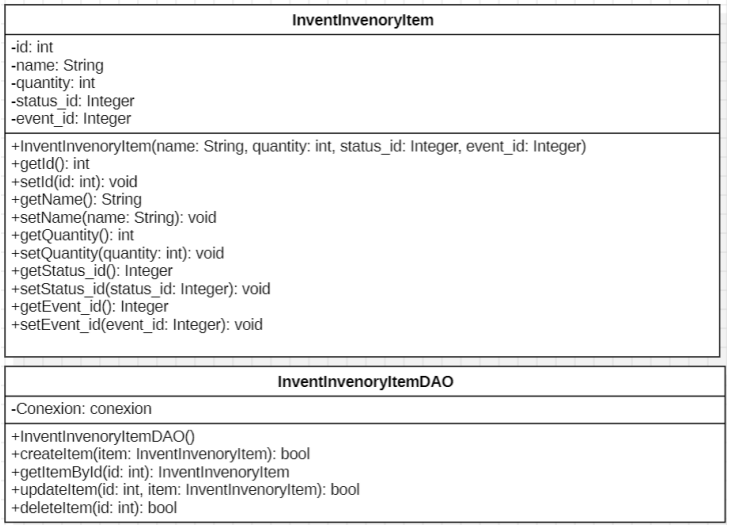

## Clases Employee y EmployeeDAO

La clase Employee define un modelo para representar a los empleados, encapsulando detalles como el nombre, la identificación, la fecha de nacimiento, el rol y el estado del empleado, permitiendo la creación y manipulación de estos datos a través de métodos getter y setter. Por otro lado, la clase EmployeeDAO maneja la interacción con la base de datos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los empleados, utilizando la clase Conexion para gestionar la conexión a la base de datos. A través de sentencias SQL preparadas, la clase EmployeeDAO se encarga de persistir y recuperar los datos de los empleados, asegurando que las operaciones se realicen de manera segura y eficiente.



## Clases InventInvenoryItem y InventInvenoryItemDAO

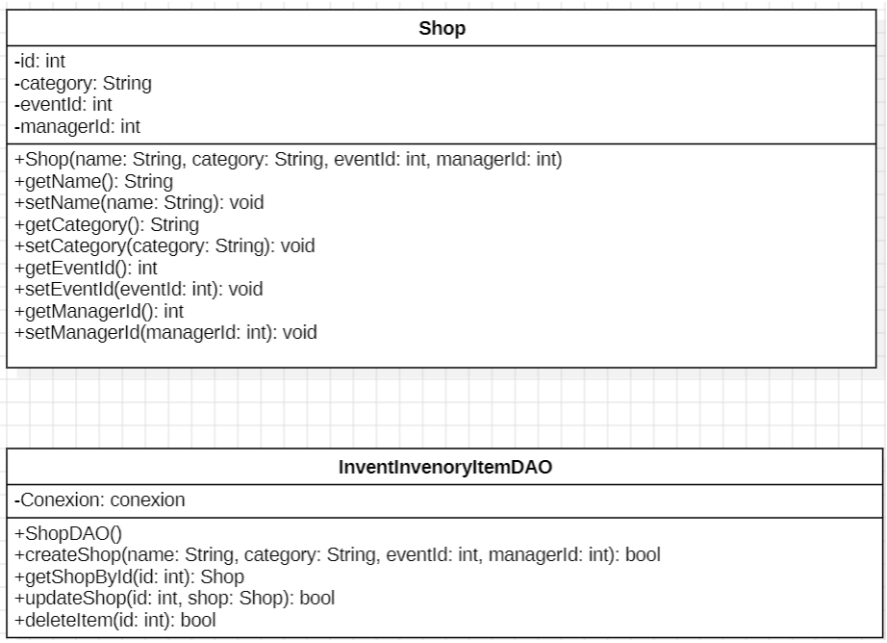
La clase InventInvenoryItem representa un ítem de inventario, encapsulando información esencial como el nombre, la cantidad, el estado y la asociación con un evento específico. Esta clase permite la creación, acceso y modificación de los atributos del ítem mediante métodos getter y setter. Por otro lado, la clase InventInvenoryItemDAO se encarga de la gestión de los ítems en la base de datos, proporcionando métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar). A través de consultas SQL preparadas, esta clase interactúa con la base de datos para asegurar la persistencia de los datos de los ítems, manejando las conexiones y posibles excepciones que puedan surgir durante la ejecución de las operaciones.




## Clases Shop y ShopDAO

La clase Shop representa un comercio asociado a un evento, encapsulando información clave como el nombre del comercio, su categoría (por ejemplo, tienda o restaurante), el identificador del evento al que está asociado, y el identificador del gerente responsable. Esta clase facilita la creación y manipulación de los datos de un comercio a través de métodos getter y setter. Por su parte, la clase ShopDAO maneja la persistencia de los datos de los comercios en la base de datos, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) mediante consultas SQL preparadas. La clase ShopDAO utiliza la clase Conexion para establecer y gestionar la conexión a la base de datos, garantizando que las

operaciones se realicen de manera segura y eficiente, y manejando cualquier excepción que pueda surgir durante el proceso.



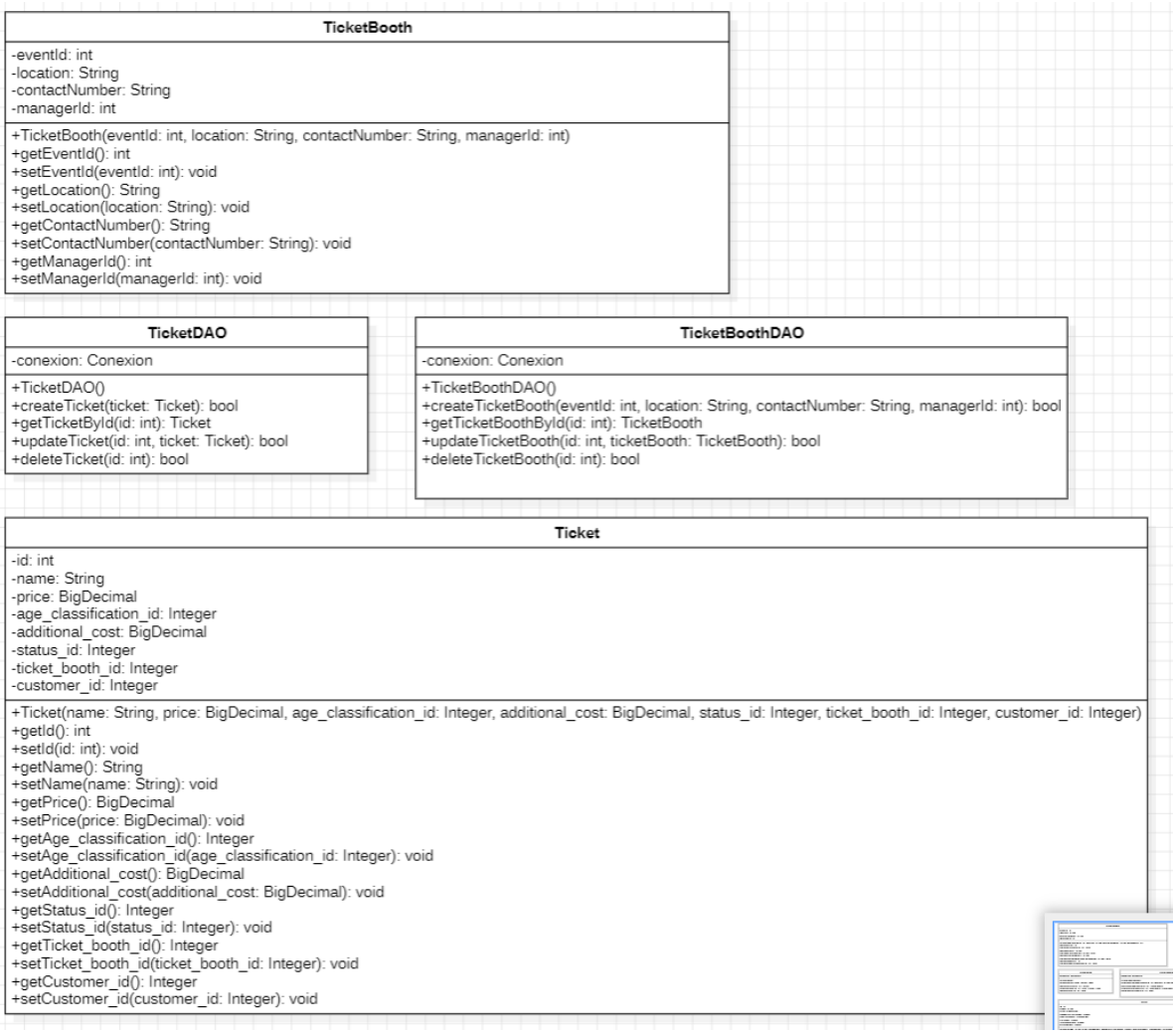

**Clases Ticket, TicketBooth, TicketDAO y TicketBoothDAO**

La clase Ticket modela un boleto para un evento, encapsulando detalles como el nombre del boleto, el precio, la clasificación por edad, el costo adicional, el estado del boleto, el identificador de la taquilla donde fue adquirido, y el identificador del cliente que lo compró. Esta clase permite acceder y modificar estos atributos a través de métodos getter y setter.

La clase TicketBooth representa una taquilla de boletos asociada a un evento específico. Contiene atributos como el identificador del evento, la ubicación de la taquilla, el número de contacto, y el identificador del gerente responsable. Los métodos getter y setter permiten gestionar estos atributos.

Por otro lado, la clase TicketDAO se encarga de la persistencia de los datos de los boletos en la base de datos, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) mediante consultas SQL preparadas. Similarmente, la clase TicketBoothDAO gestiona la persistencia de los datos de las taquillas de boletos, ofreciendo funcionalidades CRUD para la tabla TicketBooths. Ambas clases utilizan la clase Conexion para manejar las conexiones

a la base de datos y aseguran que las operaciones se realicen de manera segura, manejando las excepciones que puedan surgir durante el proceso.

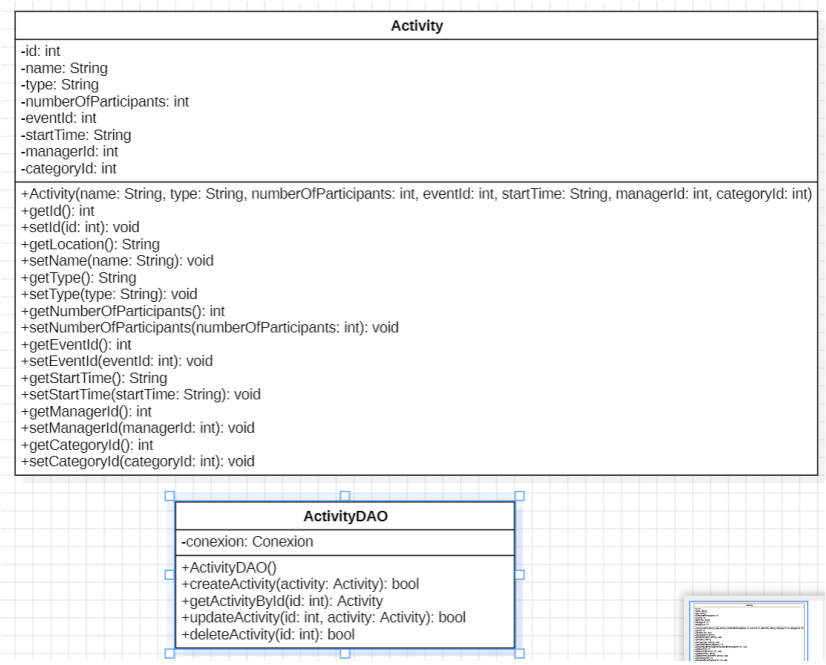



## Clases Activity y ActivityDAO

La clase Activity representa una actividad dentro de un evento, encapsulando detalles importantes como el nombre de la actividad, su tipo (por ejemplo, competencia o taller), el número de participantes, el identificador del evento al que pertenece, la hora de inicio, el identificador del gerente encargado y el identificador de la categoría a la que pertenece la actividad. Esta clase facilita la creación, acceso y modificación de los datos de una actividad a través de métodos getter y setter.

Por otro lado, la clase ActivityDAO maneja la persistencia de los datos de las actividades en la base de datos, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar)

mediante consultas SQL preparadas. Utilizando la clase Conexion, ActivityDAO gestiona la conexión a la base de datos, asegurando que las operaciones se realicen de manera segura y eficiente, y manejando las posibles excepciones que puedan surgir durante el proceso.

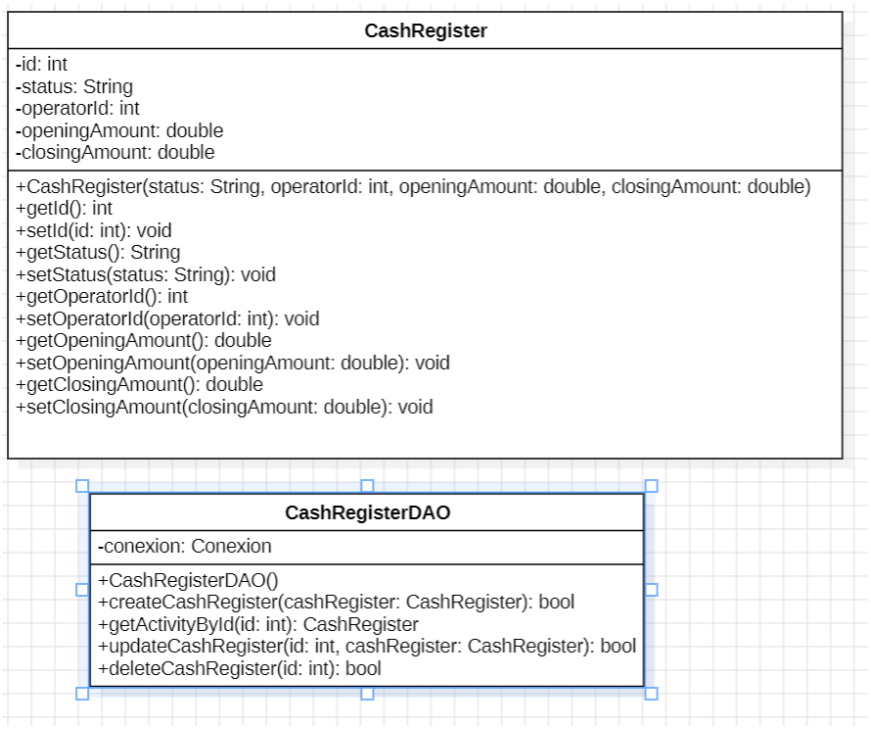



**Clases CashRegister y CashRegisterDAO**

La clase CashRegister representa una caja registradora, encapsulando detalles como el estado de la caja (por ejemplo, abierta o cerrada), el identificador del operador responsable, el monto de apertura y el monto de cierre de la caja. Esta clase facilita la creación y manipulación de los datos de una caja registradora a través de métodos getter y setter.

Por otro lado, la clase CashRegisterDAO se encarga de la persistencia de los datos de las cajas registradoras en la base de datos, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) mediante consultas SQL preparadas. Utilizando la clase Conexion, CashRegisterDAO gestiona la conexión a la base de datos, asegurando que las operaciones se realicen de manera segura y eficiente. Además, maneja cualquier excepción que pueda

surgir durante el proceso, proporcionando métodos específicos para la creación, consulta, actualización y eliminación de registros de cajas registradoras.



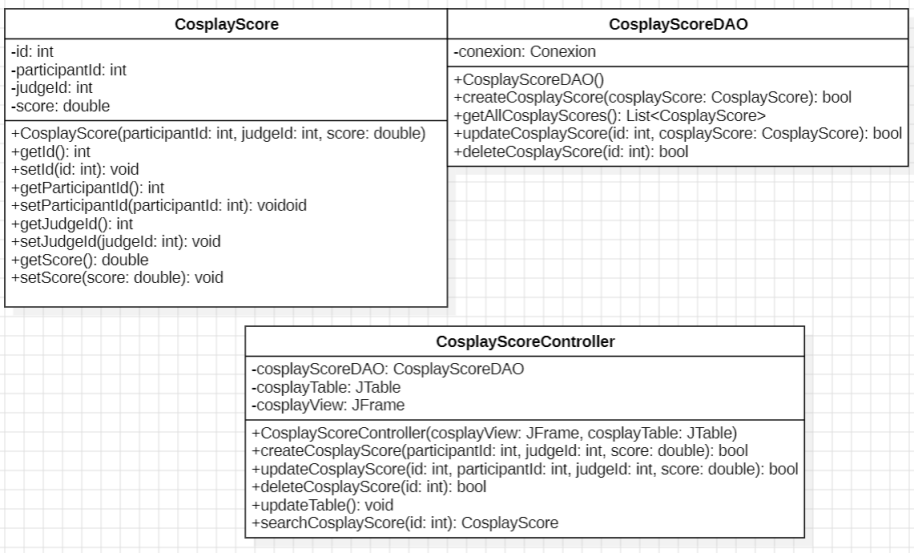

**Clases CosplayScore, CosplayScoreDAO y CosplayScoreController**

La clase CosplayScore representa un puntaje otorgado a un participante en un concurso de cosplay, capturando detalles como el identificador del participante, el identificador del juez que otorgó el puntaje, y la puntuación asignada. Esta clase permite la creación y manipulación de los datos de los puntajes de cosplay a través de métodos getter y setter.

La clase CosplayScoreDAO se encarga de la persistencia de los puntajes de cosplay en la base de datos, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) mediante consultas SQL preparadas. Utiliza la clase Conexion para manejar la conexión a la base de datos, asegurando que las operaciones se realicen de manera segura y eficiente, y manejando cualquier excepción que pueda surgir durante el proceso.

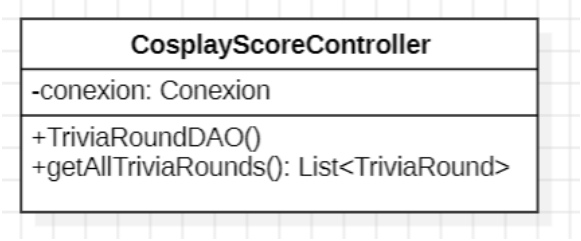
Por su parte, la clase CosplayScoreController actúa como intermediario entre la vista y el modelo, gestionando la interacción del usuario con la interfaz gráfica para manipular los puntajes de cosplay. Proporciona métodos para crear, actualizar, eliminar y buscar puntajes de cosplay, y actualiza la tabla de la interfaz con los datos más recientes. También maneja la

interacción del usuario mediante cuadros de diálogo que informan sobre el éxito o fracaso de las operaciones realizadas.




**Clase TriviaRoundDAO**

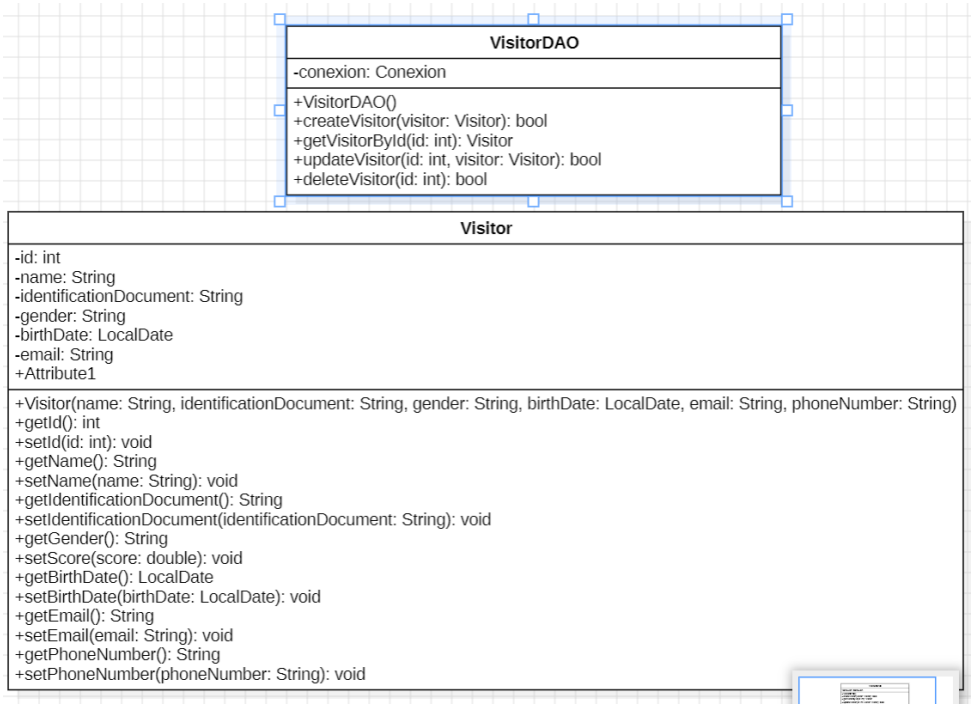
La clase TriviaRoundDAO se encarga de manejar la persistencia y recuperación de las rondas de trivia en la base de datos. A través de la conexión proporcionada por la clase Conexion, esta clase realiza consultas SQL para obtener todas las rondas de trivia almacenadas. Cada ronda de trivia se encapsula en un objeto de tipo TriviaRound, que incluye información como el identificador del evento, el número de ronda, los identificadores de los participantes, y el identificador del ganador. La clase almacena estos objetos en una lista y los devuelve al solicitante. Además, gestiona posibles excepciones que puedan surgir durante la consulta a la base de datos, asegurando una interacción segura y controlada con la base de datos.



## Clases Visitor y VisitorDAO

La clase Visitor representa un visitante, encapsulando información esencial como el nombre, el documento de identificación, el género, la fecha de nacimiento, el correo electrónico y el número de teléfono. Esta clase permite crear y manipular los datos de un visitante a través de métodos getter y setter, facilitando la gestión de la información personal del visitante.

Por otro lado, la clase VisitorDAO maneja la persistencia de los datos de los visitantes en la base de datos, permitiendo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) mediante consultas SQL preparadas. Utilizando la clase Conexion, VisitorDAO gestiona la conexión a la base de datos y se asegura de que las operaciones se realicen de manera segura y eficiente. La clase maneja posibles excepciones que puedan surgir durante el acceso a la base de datos, proporcionando métodos específicos para la creación, consulta, actualización y eliminación de registros de visitantes.

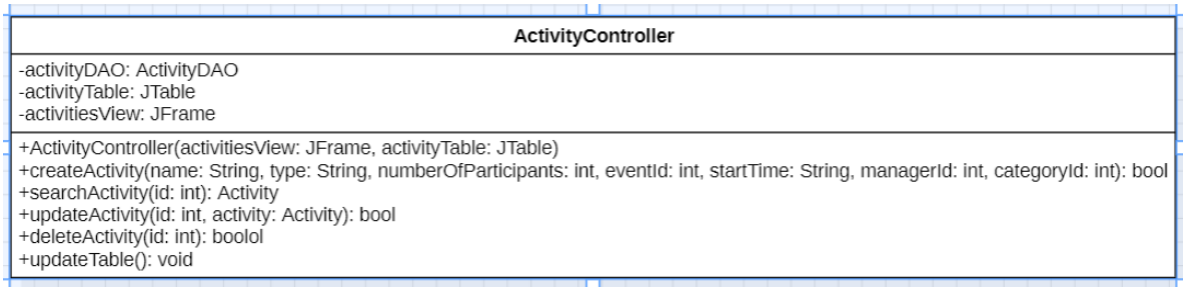



## Clase ActivityController

La clase ActivityController actúa como un intermediario entre la vista (interfaz gráfica) y el modelo, gestionando la interacción del usuario con las actividades de un evento. Utiliza la clase ActivityDAO para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar)

sobre las actividades en la base de datos. La clase también maneja la actualización de la tabla que muestra las actividades en la interfaz gráfica, cargando los datos desde la base de datos y reflejando los cambios realizados por el usuario en tiempo real.

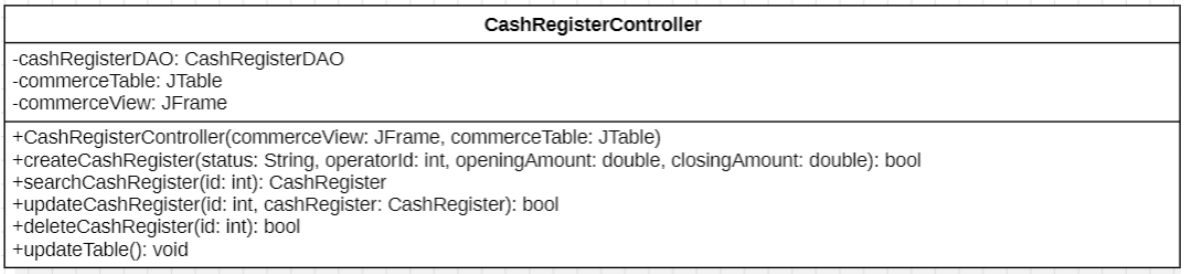
ActivityController proporciona métodos para crear nuevas actividades, buscar actividades por su identificador, actualizar actividades existentes y eliminar actividades. Además, maneja la interacción con el usuario mediante cuadros de diálogo, informando sobre el éxito o el fracaso de las operaciones realizadas. La clase garantiza que la tabla de actividades en la vista siempre esté actualizada con la información más reciente de la base de datos.




**Clase CashRegisterController**

La clase CashRegisterController es responsable de gestionar las operaciones relacionadas con las cajas registradoras en una aplicación, actuando como un puente entre la interfaz gráfica y la lógica de negocios. Esta clase utiliza CashRegisterDAO para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos. Proporciona métodos para crear nuevas cajas registradoras, buscar una caja registradora específica por su ID, actualizar la información de una caja existente y eliminar cajas registradoras.

Además, CashRegisterController maneja la actualización de la tabla que muestra las cajas registradoras en la interfaz gráfica (JTable), asegurando que los datos mostrados siempre estén sincronizados con la base de datos. La clase también maneja la interacción del usuario mediante cuadros de diálogo, informando sobre el éxito o fracaso de las operaciones realizadas. Maneja excepciones SQL para garantizar que cualquier problema durante la operación se gestione de manera adecuada, proporcionando mensajes de error claros al usuario.

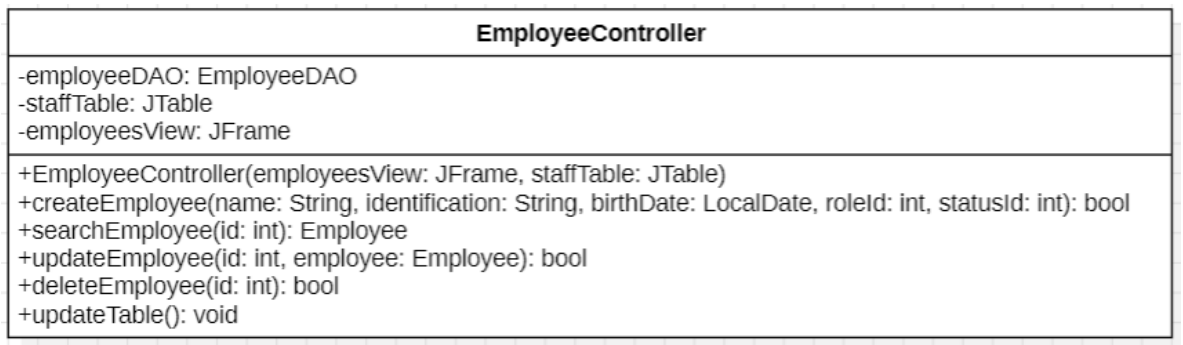



## Clase EmployeeController

La clase EmployeeController es responsable de gestionar las operaciones relacionadas con los empleados dentro de una aplicación. Actúa como un intermediario entre la vista (interfaz gráfica) y la lógica de negocio, utilizando la clase EmployeeDAO para interactuar con la base de datos. Esta clase proporciona métodos para crear, buscar, actualizar y eliminar empleados, garantizando que las operaciones se reflejen en la base de datos y se actualicen en la interfaz gráfica.

El método createEmployee permite crear un nuevo empleado con los datos proporcionados, mientras que searchEmployee busca un empleado por su ID. El método updateEmployee actualiza la información de un empleado existente, y deleteEmployee elimina un empleado de la base de datos. Además, updateTable se encarga de actualizar la tabla de empleados en la interfaz gráfica, mostrando los datos más recientes de la base de datos.

La clase también maneja excepciones SQL para asegurarse de que cualquier problema durante la operación sea gestionado adecuadamente, proporcionando mensajes de error claros al usuario a través de cuadros de diálogo. Además, la clase garantiza que los recursos de la base de datos, como conexiones y resultados, se cierren correctamente después de cada operación.



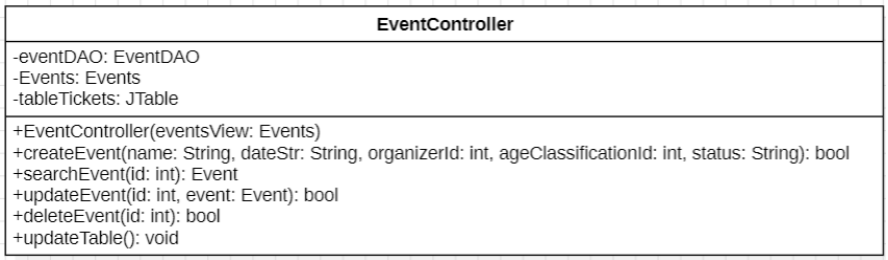
## Clase EventController

La clase EventController se encarga de gestionar las operaciones relacionadas con los eventos dentro de una aplicación. Actúa como un intermediario entre la vista, representada por la clase Events, y la lógica de negocio, utilizando la clase EventDAO para interactuar con la base de datos. Esta clase proporciona métodos para crear, buscar, actualizar y eliminar eventos, garantizando que las operaciones se reflejen en la base de datos y se actualicen en la interfaz gráfica.

El método createEvent permite crear un nuevo evento, asegurándose de que la fecha proporcionada esté en el formato correcto (yyyy-MM-dd HH:mm

). Si el formato es incorrecto, se muestra un mensaje de error al usuario. Los métodos searchEvent, updateEvent y deleteEvent permiten buscar un evento por su ID, actualizar la información de un evento existente y eliminar un evento, respectivamente. Además, tras la creación o modificación de un evento, se actualiza la tabla en la vista para reflejar los cambios realizados.

La clase maneja excepciones, mostrando mensajes de error claros al usuario a través de cuadros de diálogo cuando se produce un problema durante la operación.

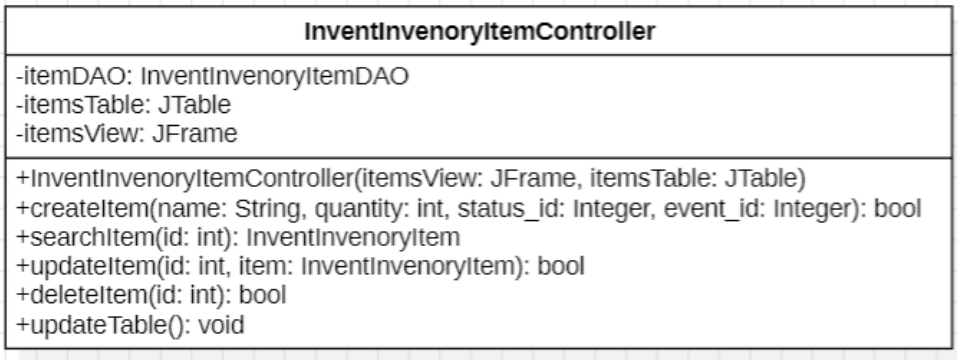



## Clase InventInvenoryItemController

La clase InventInvenoryItemController es responsable de gestionar las operaciones relacionadas con los ítems de inventario en una aplicación. Esta clase actúa como un intermediario entre la vista (interfaz gráfica) y la lógica de negocios, utilizando la clase InventInvenoryItemDAO para interactuar con la base de datos. Proporciona métodos para crear, buscar, actualizar y eliminar ítems de inventario, asegurando que las operaciones se reflejen en la base de datos y se actualicen en la interfaz gráfica.

El método createItem permite crear un nuevo ítem en el inventario, con los detalles proporcionados como nombre, cantidad, estado y evento asociado. Los métodos searchItem, updateItem y deleteItem permiten buscar un ítem por su ID, actualizar la información de un ítem existente y eliminar un ítem, respectivamente. Además, el método updateTable se encarga de actualizar la tabla de ítems en la interfaz gráfica, cargando los datos más recientes desde la base de datos.

La clase también maneja excepciones SQL para garantizar que cualquier problema durante la operación sea gestionado adecuadamente, proporcionando mensajes de error claros al usuario a través de cuadros de diálogo. Se asegura de que los recursos de la base de datos, como conexiones y resultados, se cierren correctamente después de cada operación.

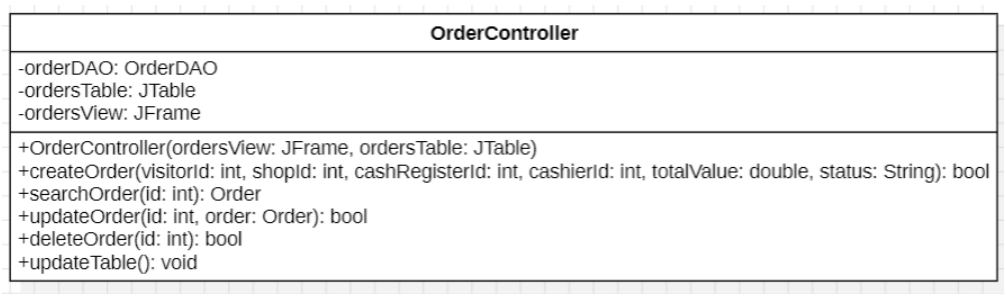



**Clase OrderController**

La clase OrderController es responsable de gestionar las operaciones relacionadas con los pedidos dentro de una aplicación. Actúa como un intermediario entre la vista (interfaz gráfica) y la lógica de negocios, utilizando la clase OrderDAO para interactuar con la base de datos. Proporciona métodos para crear, buscar, actualizar y eliminar pedidos, garantizando que las operaciones se reflejen en la base de datos y se actualicen en la interfaz gráfica.

El método createOrder permite crear un nuevo pedido con detalles como el ID del visitante, ID de la tienda, ID de la caja registradora, ID del cajero, valor total y estado del pedido. Los métodos searchOrder, updateOrder y deleteOrder permiten buscar un pedido por su ID, actualizar la información de un pedido existente y eliminar un pedido, respectivamente. Además, el método updateTable se encarga de actualizar la tabla de pedidos en la interfaz gráfica, cargando los datos más recientes desde la base de datos.

La clase maneja excepciones SQL para asegurarse de que cualquier problema durante la operación sea gestionado adecuadamente, proporcionando mensajes de error claros al usuario a través de cuadros de diálogo. También se asegura de que los recursos de la base de datos, como conexiones y resultados, se cierren correctamente después de cada operación.

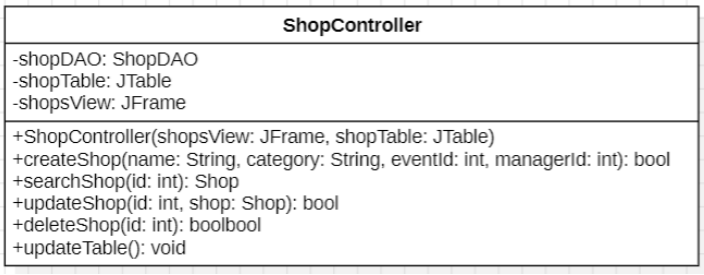



**Clase ShopController**

La clase ShopController es responsable de gestionar las operaciones relacionadas con los comercios (tiendas y restaurantes) dentro de una aplicación. Actúa como un intermediario entre la vista (interfaz gráfica) y la lógica de negocios, utilizando la clase ShopDAO para interactuar con la base de datos. Proporciona métodos para crear, buscar, actualizar y eliminar comercios, garantizando que las operaciones se reflejen en la base de datos y se actualicen en la interfaz gráfica.

El método createShop permite crear un nuevo comercio con detalles como el nombre, la categoría (por ejemplo, tienda o restaurante), el ID del evento asociado y el ID del gerente. Los métodos searchShop, updateShop y deleteShop permiten buscar un comercio por su ID, actualizar la información de un comercio existente y eliminar un comercio, respectivamente. Además, el método updateTable se encarga de actualizar la tabla de comercios en la interfaz gráfica, cargando los datos más recientes desde la base de datos.

La clase maneja excepciones SQL para asegurarse de que cualquier problema durante la operación sea gestionado adecuadamente, proporcionando mensajes de error claros al usuario a través de cuadros de diálogo. También se asegura de que los recursos de la base de datos, como conexiones y resultados, se cierren correctamente después de cada operación.

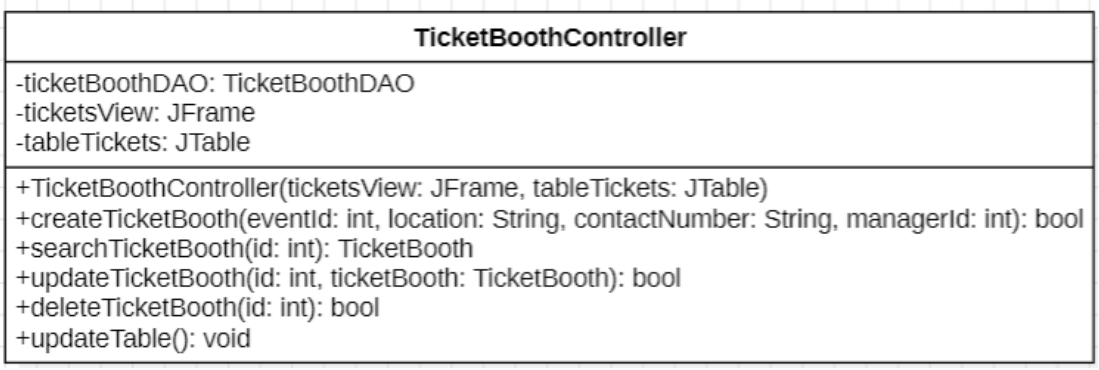



**Clase TicketBoothController**

La clase TicketBoothController es responsable de gestionar las operaciones relacionadas con las taquillas dentro de una aplicación. Actúa como un intermediario entre la vista (interfaz gráfica) y la lógica de negocios, utilizando la clase TicketBoothDAO para interactuar con la base de datos. Proporciona métodos para crear, buscar, actualizar y eliminar taquillas, asegurando que las operaciones se reflejen en la base de datos y se actualicen en la interfaz gráfica.

El método createTicketBooth permite crear una nueva taquilla con detalles como el ID del evento, la ubicación, el número de contacto y el ID del gerente. Los métodos searchTicketBooth, updateTicketBooth y deleteTicketBooth permiten buscar una taquilla por su ID, actualizar la información de una taquilla existente y eliminar una taquilla, respectivamente. Además, el método updateTable se encarga de actualizar la tabla de taquillas en la interfaz gráfica, cargando los datos más recientes desde la base de datos.

La clase maneja excepciones SQL para asegurarse de que cualquier problema durante la operación sea gestionado adecuadamente, proporcionando mensajes de error claros al usuario a través de cuadros de diálogo. También se asegura de que los recursos de la base de datos, como conexiones y resultados, se cierren correctamente después de cada operación.



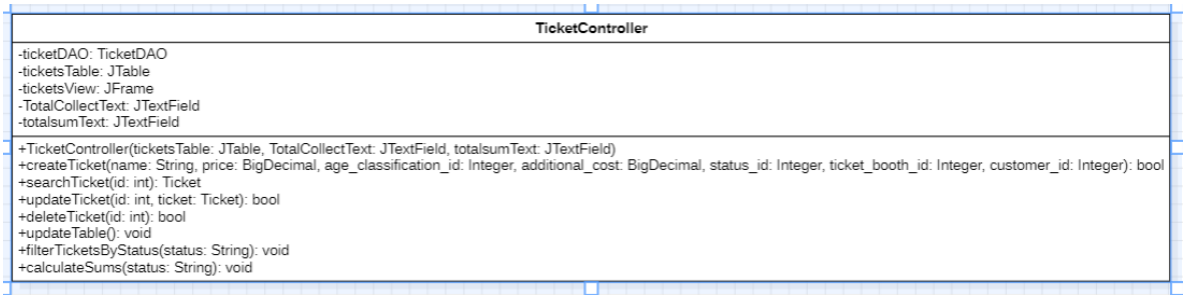
## Clase TicketController

La clase TicketController es responsable de gestionar las operaciones relacionadas con los tickets dentro de una aplicación. Actúa como un intermediario entre la vista (interfaz gráfica) y la lógica de negocios, utilizando la clase TicketDAO para interactuar con la base de datos. Proporciona métodos para crear, buscar, actualizar y eliminar tickets, así como filtrar y calcular sumas relacionadas con los tickets según su estado.

El método createTicket permite crear un nuevo ticket con detalles como nombre, precio, clasificación por edad, costo adicional, estado, ID de la taquilla e ID del cliente. Los métodos searchTicket, updateTicket y deleteTicket permiten buscar un ticket por su ID, actualizar la información de un ticket existente y eliminar un ticket, respectivamente. Además, el método updateTable se encarga de actualizar la tabla de tickets en la interfaz gráfica, cargando los datos más recientes desde la base de datos.

El método filterTicketsByStatus permite filtrar los tickets en la tabla según su estado, mientras que el método calculateSums calcula y muestra el total recaudado y el número total de tickets según el estado filtrado.

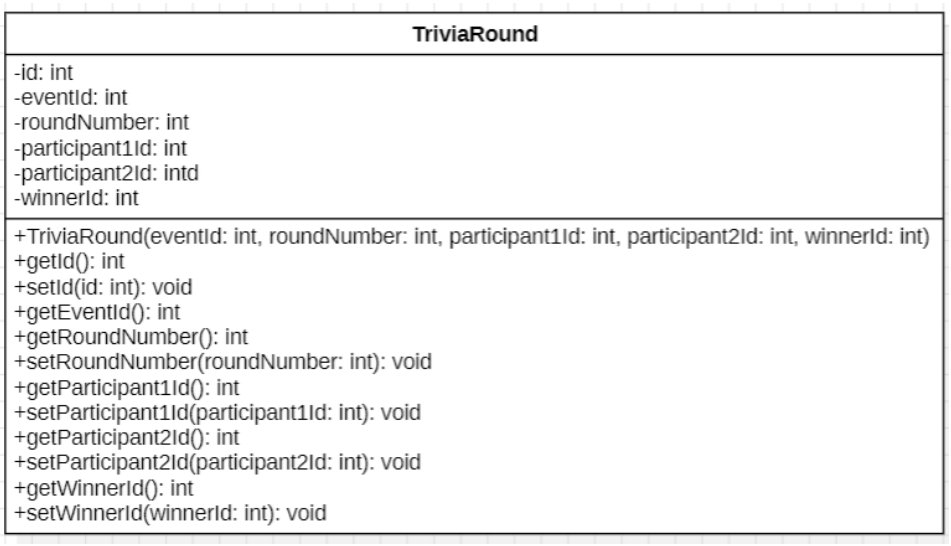
La clase maneja excepciones SQL para asegurarse de que cualquier problema durante la operación sea gestionado adecuadamente, proporcionando mensajes de error claros al usuario a través de cuadros de diálogo. También se asegura de que los recursos de la base de datos, como conexiones y resultados, se cierren correctamente después de cada operación.




## Clase TriviaRound

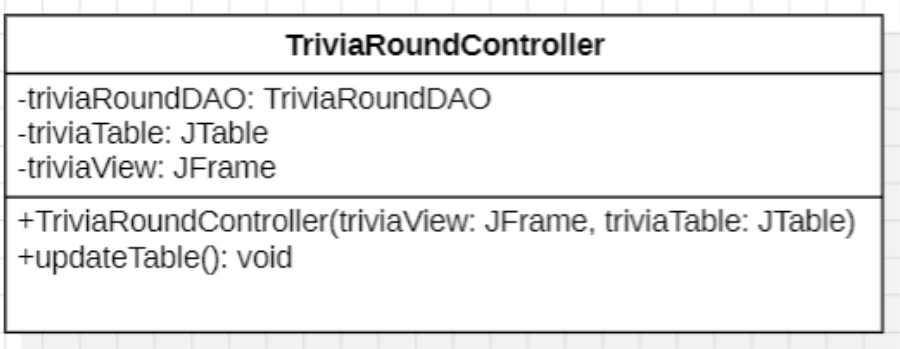
La clase TriviaRound es un modelo que representa una ronda de trivia en un evento, gestionando detalles como el ID del evento, el número de la ronda, los IDs de los dos participantes y el ID del ganador. A través de sus atributos y métodos, se permite almacenar

y acceder a esta información, facilitando el seguimiento de los participantes y los resultados de cada ronda dentro de un evento específico. Los métodos get y set permiten manipular cada uno de estos atributos, asegurando que la información esté siempre actualizada y accesible para otras partes del sistema.



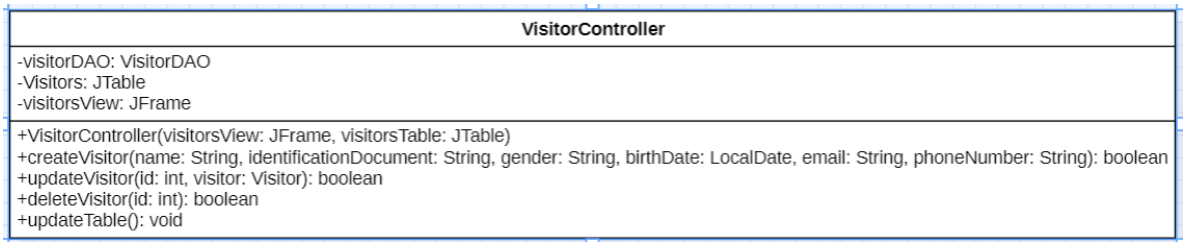

**Clase TriviaRoundController**

La clase TriviaRoundController actúa como un intermediario entre la interfaz gráfica y la lógica de negocios para gestionar las rondas de trivia, utilizando TriviaRoundDAO para acceder a la base de datos. A través de su método updateTable, la clase actualiza la tabla de la interfaz, limpiándola y cargando los datos más recientes de las rondas de trivia, mostrando cada ronda como una nueva fila en la tabla. Los atributos incluyen una instancia de TriviaRoundDAO, la tabla gráfica triviaTable y la ventana JFrame triviaView, que contiene la vista de trivia.



**Clase VisitorController**

La clase VisitorController actúa como un puente entre la vista y la lógica de negocio relacionada con la gestión de visitantes, utilizando la clase VisitorDAO para realizar operaciones en la base de datos. Proporciona métodos para crear, buscar, actualizar y eliminar registros de visitantes, además de actualizar la tabla gráfica que muestra los datos de los visitantes. La clase contiene un método llamado updateTable que limpia y recarga la tabla con la información más reciente de los visitantes desde la base de datos. Los atributos principales incluyen una instancia de VisitorDAO, la tabla gráfica llamada Visitors y la ventana JFrame visitorsView, que representa la interfaz gráfica donde se visualizan y gestionan los visitantes.




# Conclusiones y Opciones de Mejora

El desarrollo del software de gestión de eventos de anime ha permitido establecer una base sólida para la administración eficiente de eventos, incluyendo la gestión de actividades, comercios, boletos y personal involucrado. A lo largo del proyecto, se han implementado diversas funcionalidades críticas que garantizan un control exhaustivo de los recursos y la optimización de los procesos operativos. Sin embargo, existen varias áreas en las que se pueden realizar mejoras para fortalecer aún más la eficiencia, escalabilidad y flexibilidad del sistema:

1. **Optimización de Consultas y Rendimiento de la Base de Datos:** A medida que el sistema maneja un volumen creciente de datos, es fundamental optimizar las consultas SQL para mejorar el tiempo de respuesta. La creación de índices adicionales en columnas frecuentemente consultadas, así como la implementación de técnicas de partición de tablas, podría reducir la carga en la base de datos y mejorar el rendimiento general del sistema.

2. **Ampliación de Funcionalidades de Reportes:** Actualmente, el sistema genera reportes financieros y de actividades, pero se podrían incluir opciones más avanzadas de análisis de datos, como reportes personalizados que permitan a los organizadores realizar un seguimiento más detallado del comportamiento de los asistentes, tendencias de ventas, y efectividad de las promociones durante los eventos.

3. **Integración de Módulos de Seguridad Avanzada:** Aunque el sistema incluye un manejo básico de roles y permisos, se podría fortalecer la seguridad mediante la implementación de autenticación multifactor (MFA) y encriptación de datos sensibles. Además, la auditoría de acceso y la trazabilidad de cambios en la base de datos proporcionarían un mayor control sobre el acceso a la información y la modificación de registros.

4. **Mejora en la Interfaz de Usuario (UI) y Experiencia de Usuario (UX):** La usabilidad del sistema puede mejorarse mediante una interfaz más intuitiva y atractiva visualmente. Se podrían incorporar elementos de diseño más modernos y accesibles, así como optimizaciones para dispositivos móviles, lo cual permitiría a los organizadores y asistentes interactuar con el sistema de manera más eficiente en diferentes plataformas.

5. **Automatización y Escalabilidad del Sistema:** Para facilitar la administración de múltiples eventos simultáneos, sería conveniente integrar funcionalidades de automatización, como la asignación automática de roles y la generación de itinerarios de eventos basados en plantillas predefinidas. Además, la adopción de tecnologías de microservicios podría facilitar la escalabilidad del sistema, permitiendo la expansión de funcionalidades sin afectar el rendimiento.

6. **Incorporación de Inteligencia Artificial (IA):** Integrar módulos de inteligencia artificial podría añadir valor significativo al sistema, permitiendo prever la demanda de boletos, optimizar la distribución del personal en función del flujo de asistentes, y recomendar productos o actividades basadas en las preferencias de los visitantes.

7. **Monitoreo y Mantenimiento Continuo:** Establecer un sistema de monitoreo continuo que permita identificar y corregir problemas en tiempo real garantizará la

estabilidad del sistema durante los eventos. La implementación de un plan de mantenimiento proactivo, con revisiones periódicas y actualizaciones de seguridad, ayudará a mantener el software actualizado y protegido contra vulnerabilidades emergentes.

# Referencias


- Oracle. (2023). *JDBC Basics*. Oracle Documentation. https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html
- JavaFX. (2023). *Using JavaFX UI Controls*. https://openjfx.io/javadoc/17/javafx.controls/javafx/scene/control/package-summary.html
- Oracle. (s.f.). MySQL Workbench: Guía del usuario. Recuperado de https://dev.mysql.com/doc/workbench/en
- Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). Database Systems: The Complete Book (2nd ed.). Pearson Education