ArteVida Cultural

Este proyecto consiste en crear una base de datos para gestionar eventos culturales, registrando actividades, artistas, ubicaciones y asistentes, con el objetivo de documentar, consultar y analizar la información, utilizando un modelo entidad-relación y MySQL para su implementación.

Bases de datos SQL | Bloque 2

Pablo Fernández Lucas

Índice general

1. Diseño conceptual

A partir de los requisitos del problema propuesto se construirá el modelo conceptual de datos, en concreto el modelo entidad-relación.

1.1 Contexto

La empresa .ArteVida Cultural.Organiza eventos culturales diversos como conciertos, exposiciones, obras de teatro y conferencias, con el objetivo de llevar la cultura a diferentes comunidades. Cada evento tiene un nombre único, se realiza en una ubicación específica con capacidad y costos asociados, y puede incluir uno o varios artistas, cuyos cachés varían según el tipo de actividad. Además, la empresa gestiona la venta de entradas y mantiene un registro de los asistentes, permitiendo consultas sobre tipos de eventos, localización y popularidad en fechas específicas para optimizar decisiones estratégicas.

1.2 Entidades

Bajo este contexto identificamos las siguientes entidades, que más tarden formarán parte de nuestro diagrama de Entidad-Relación.

1. **Actividad:** Representa las actividades culturales organizadas. Incluye nombre, tipo (concierto, exposición, teatro, etc.) y coste. Es esencial para categorizar los eventos y conocer el coste asociado, facilitando el análisis y la planificación.
2. **Artista:** Almacena información sobre los artistas, como nombre, caché (dependiendo del evento) y biografía. Permite registrar datos, definir la participación en actividades y calcular el coste total del evento.
3. **Ubicación:** Representa el lugar del evento, con atributos como nombre, dirección, ciudad/pueblo, aforo, precio de alquiler y características. Facilita la gestión logística y permite consultas relacionadas con las ubicaciones.
4. **Evento:** Contiene información específica de cada evento: nombre, actividad, ubicación, precio, fecha, hora y descripción. Es clave para registrar eventos y realizar consultas analíticas.
5. **Asistente:** Almacena datos de los asistentes, como nombre completo, teléfonos y email. Permite gestionar la audiencia y hacer análisis sobre la participación.

1.3 Diagrama Entidad - Relación

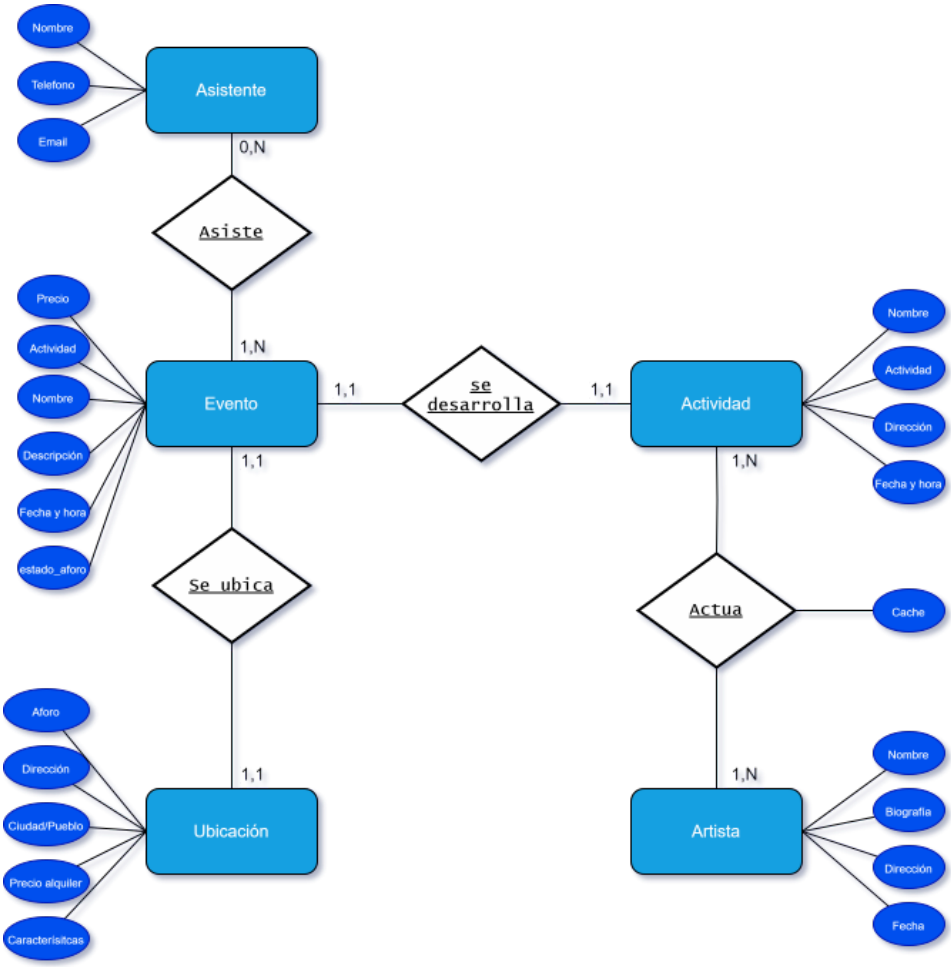


Figura 1.1: Diagrama Entidad-Relación

1.4 Entidades y sus atributos

Nota: Se han incluido los identificadores y las claves foráneas en este apartado para detallar sus respectivas restricciones.

1.4.1 ASISTENTE

La entidad **ASISTENTE** representa a las personas que asisten a los eventos. Contiene los siguientes atributos:

- **id_asistente**: Es un identificador único para cada asistente. Su valor es de tipo entero

y no puede estar vacío, ya que es la clave primaria de la tabla.

- **nombre**: Almacena el nombre completo del asistente. Es un texto de hasta 100 caracteres y no puede estar vacío, ya que es un dato esencial.
- **telefono**: Guarda el número de teléfono del asistente. Es un texto corto (hasta 15 caracteres), pero este dato es opcional.
- **email**: Representa la dirección de correo electrónico del asistente. Es un texto de hasta 100 caracteres, debe ser único en la tabla y seguir un formato válido (como usuario@dominio.com). Este campo es opcional.

Restricciones:

- El identificador **id_asistente** debe ser único y no nulo.
- El correo electrónico, si se proporciona, debe ser único y tener un formato válido.
- El número de teléfono, si se proporciona, debe cumplir un formato válido para evitar inconsistencias.

1.4.2 EVENTO

La entidad **EVENTO** almacena información sobre los eventos organizados. Contiene los siguientes atributos:

- **id_evento**: Es un identificador único para cada evento. Es de tipo entero y no puede estar vacío, ya que actúa como clave primaria.
- **nombre**: Representa el nombre del evento. Es un texto de hasta 150 caracteres y no puede estar vacío.
- **descripcion**: Describe el evento con mayor detalle. Es un texto de longitud variable y puede ser opcional.
- **fecha_hora**: Indica la fecha y la hora en que se realizará el evento. Este atributo es obligatorio y debe ser válido según el formato de fecha y hora.
- **precio**: Es el costo de asistencia al evento. Es un número decimal que debe ser mayor o igual a cero, y su registro es obligatorio.
- **id_ubicacion**: Representa una relación con la entidad **UBICACION**. Es una clave foránea que conecta un evento con una ubicación específica. Este valor no puede estar vacío.

- **id_actividad**: Representa una relación con la entidad **ACTIVIDAD**. Es una clave foránea que conecta un evento con la actividad que se llevará a cabo. Este valor tampoco puede estar vacío.
- **estado_aforo**: Este atributo es de tipo booleano (con valores 0 o 1) y su propósito es reflejar de forma rápida y eficiente el estado del aforo de un evento. Los posibles valores son los siguientes:
  - **1**: Indica que el aforo está completo.
  - **0**: Indica que el aforo no está completo.

> **Nota:** Aunque el valor de este atributo podría calcularse dinámicamente mediante una vista, hemos optado por gestionarlo a través de un trigger. Este trigger se encargará de actualizar el valor de **estado_aforo** cada vez que se registre un nuevo asistente, verificando si el aforo del evento ha alcanzado su límite o no.

Restricciones:

- **id_evento** debe ser único y no nulo.
- El **precio** debe ser mayor o igual a cero.
- **id_ubicacion** e **id_actividad** deben corresponder a registros existentes en las entidades relacionadas.

1.4.3 ACTIVIDAD

La entidad **ACTIVIDAD** detalla las actividades realizadas en los eventos. Contiene los siguientes atributos:

- **id_actividad**: Es un identificador único para cada actividad. Es de tipo entero y no puede estar vacío, ya que es la clave primaria de la tabla.
- **nombre**: Representa el nombre de la actividad. Es un texto de hasta 150 caracteres y no puede estar vacío.
- **direccion**: Describe la dirección donde se realiza la actividad. Es un texto de hasta 200 caracteres y es opcional.
- **fecha_hora**: Indica la fecha y la hora de la actividad. Este atributo es obligatorio y debe ser válido según el formato de fecha.

Restricciones:

- El identificador **id_actividad** debe ser único y no nulo.
- El atributo **fecha** debe seguir un formato válido y no puede estar vacío.

1.4.4 ARTISTA

La entidad **ARTISTA** representa a las personas que participan en las actividades. Contiene los siguientes atributos:

- **id_artista**: Es un identificador único para cada artista. Es de tipo entero y no puede estar vacío, ya que es la clave primaria de la tabla.
- **nombre**: Almacena el nombre del artista. Es un texto de hasta 150 caracteres y no puede estar vacío.
- **biografia**: Describe detalles del artista. Es un texto de longitud variable y es opcional.

Nota: Dado que el **caché** del artista depende de la actividad que realice, no lo añadiremos como atributo de la entidad, sino como atributo de la relación **ACTUA** entre artista y actividad.

Restricciones:

- El identificador **id_artista** debe ser único y no nulo.
- El atributo **cache** debe ser mayor o igual a cero en la relación **ACTUA**.

1.4.5 UBICACION

La entidad **UBICACION** representa los lugares donde se realizan los eventos. Contiene los siguientes atributos:

- **id_ubicacion**: Es un identificador único para cada ubicación. Es de tipo entero y no puede estar vacío, ya que es la clave primaria de la tabla.
- **aforo**: Indica la capacidad máxima del lugar. Es un número entero que debe ser mayor o igual a 1 y no puede estar vacío.

- **direccion**: Representa la dirección del lugar. Es un texto de hasta 200 caracteres y no puede estar vacío.
- **ciudad_pueblo**: Indica la localidad donde se encuentra la ubicación. Es un texto de hasta 100 caracteres y no puede estar vacío.
- **precio_alquiler**: Representa el costo de alquiler del lugar. Es un número decimal que debe ser mayor o igual a cero, y su registro es obligatorio.
- **caracteristicas**: Describe las características adicionales del lugar. Es un texto opcional.

Restricciones:

- El identificador **id_ubicacion** debe ser único y no nulo.
- El atributo **aforo** debe ser un número mayor o igual a 1.
- El atributo **precio_alquiler** debe ser mayor o igual a cero.

2. Modelo Relacional

2.1 Paso a tablas

- **ASISTENTE** (id_asistente, nombre, telefono, email)
- **EVENTO** (id_evento, nombre, descripcion, fecha_hora, precio, id_ubicacion, id_actividad, estado_aforo)
- **ACTIVIDAD** (id_actividad, nombre, direccion, fecha_hora)
- **ARTISTA** (id_artista, nombre, biografia)
- **UBICACION** (id_ubicacion, aforo, direccion, ciudad_pueblo, precio_alquiler, caracteristicas)
- **ASISTE**(id_evento, id_asistente)
- **ACTUA**(id_actividad, id_artista, caché)

2.2 Relaciones

2.2.1 Relaciones 1-N

- EVENTO $\xrightarrow{id\_ubicacion}$ UBICACION
- EVENTO $\xrightarrow{id\_actividad}$ ACTIVIDAD
- ASISTE $\xrightarrow{id\_evento}$ EVENTO
- ASISTE $\xrightarrow{id\_asistente}$ ASISTENTE
- ACTUA $\xrightarrow{id\_artista}$ ARTISTA
- ACTUA $\xrightarrow{id\_actividad}$ ACTIVIDAD

2.3 Restricciones de Borrado y Edición

En la base de datos diseñada, es necesario definir políticas claras para el manejo de restricciones de borrado y edición en las entidades y relaciones. Estas restricciones aseguran la integridad de los datos y evitan inconsistencias en el modelo.

2.3.1 Restricciones de Borrado

- **ASISTENTE:** Un asistente puede ser eliminado de la base de datos, y en caso de tener registros asociados en la tabla **ASISTE**, se aplicará una política de **cascada** (*CASCADE*). Esto significa que al eliminar al asistente, todos los registros relacionados en la tabla **ASISTE** también serán eliminados automáticamente.
- **EVENTO:** Un evento no puede ser eliminado si existen registros asociados en las tablas **ASISTE** o **ACTUA**. Se aplicará una política de **restricción** para evitar eliminar eventos que aún tengan asistentes registrados o artistas asignados.
- **UBICACION:** Una ubicación no puede ser eliminada si está siendo utilizada en algún evento. En este caso, también se aplicará una política de **restricción**. Esto asegura que la eliminación de una ubicación no afecte eventos programados.
- **ACTIVIDAD:** Una actividad no puede ser eliminada si está asociada a algún evento. La política será de **restricción**, y la eliminación solo será posible si no existen referencias en la tabla **EVENTO**.
- **ARTISTA:** Un artista podrá ser eliminado en cualquier momento; en tal caso, se propagará en forma de cascada a la tabla **ACTUA** para proceder a eliminar su registro.

> **Posible mejora:** No sería ideal tener un evento con una actividad ausente de artistas; por lo tanto, restringir que el número de artistas sea mayor o igual a 1 podría ser una regla para todo evento. De esta manera, un artista no podrá abandonar una actividad ni ser eliminado si es el último artista que queda. En tal caso, deberá ser sustituido por otro artista.

2.3.2 Restricciones de Edición

- **ASISTENTE:** Los datos de un asistente, como su nombre, teléfono o email, pueden ser editados libremente, ya que no afectan la integridad de la base de datos. Sin embargo, el atributo **id_asistente** no puede ser modificado, ya que es la clave primaria

y está relacionado con la tabla **ASISTE**. Las actualizaciones que afecten las claves foráneas asociadas se realizarán automáticamente en cascada, garantizando la consistencia referencial.

- **EVENTO:** Los datos de un evento, como su nombre, descripción, fecha_hora, estado_aforo o precio, pueden ser modificados siempre que no alteren su integridad referencial. Los atributos **id_ubicacion** e **id_actividad**, que son claves foráneas, solo pueden ser editados si las nuevas referencias son válidas en las tablas **UBICACION** y **ACTIVIDAD**. Las actualizaciones en estas referencias se propagarán automáticamente en cascada a todas las tablas relacionadas.

- **UBICACION:** Los datos de una ubicación, como su dirección, aforo o características, pueden ser editados sin restricciones, siempre y cuando no se viole la regla de que el **aforo** debe ser mayor o igual a 1. El atributo **id_ubicacion**, al ser clave primaria y foránea en **EVENTO**, no puede ser modificado directamente. Sin embargo, las actualizaciones válidas se propagarán en cascada a las tablas que dependan de esta clave.

- **ACTIVIDAD:** Los datos de una actividad, como su nombre, dirección o fecha, pueden ser editados siempre que no se modifique el atributo **id_actividad**, ya que es clave primaria y foránea en **EVENTO** y **ACTUA**. Las modificaciones relacionadas con las claves foráneas serán actualizadas automáticamente en cascada, respetando la integridad referencial.

- **ARTISTA:** Los datos de un artista, como su nombre, biografía o caché, pueden ser editados sin restricciones significativas. El atributo **id_artista**, al ser clave primaria y foránea en **ACTUA**, no puede ser modificado directamente. No obstante, las actualizaciones permitidas serán propagadas en cascada para mantener la consistencia de los datos en todas las tablas relacionadas.

2.4 Triggers de Integridad y Automatización

En esta sección se detallan los automatismos que serán implementados posteriormente en la etapa de desarrollo. Estos mecanismos, conocidos como triggers, tienen como objetivo garantizar la integridad de la información, automatizar tareas clave y simplificar la gestión de los datos, permitiendo al sistema operar de forma más eficiente y confiable.

2.4.1 Triggers para Validación de Datos

- **Validación del Aforo en UBICACION:** Este trigger asegura que el número de asistentes en un evento no supere el aforo máximo permitido de la ubicación asignada.

  Activación: Se ejecuta **antes** de insertar o eliminar un registro en la tabla **ASISTE**.

  Acciones:

  - Calcula el número total de asistentes registrados y lo compara con el atributo *aforo* de la tabla **UBICACION**.
  - Actualiza el atributo *estado_aforo* en la tabla **EVENTO** a:
    - **1** si el aforo está completo.
    - **0** si aún hay capacidad disponible.
  - Si el aforo está completo, rechaza nuevos registros en **ASISTE** con un mensaje de error.

- **Validación del Formato de Correo y Teléfono en ASISTENTE:** Este trigger valida que el formato de los datos de contacto de un asistente sea correcto al momento de su registro.

  Activación: Se ejecuta **antes** de insertar un registro en la tabla **ASISTENTE**.

  Acciones:

  - Verifica que *email* cumpla con un formato válido (por ejemplo, *usuario@dominio.com*).
  - Verifica que *telefono* contenga solo caracteres numéricos o símbolos permitidos (como *+34-123-456-789*).
  - Si algún dato no cumple con el formato, cancela la operación y muestra un mensaje de error.

- **Validación de Fechas en EVENTO y ACTIVIDAD:** Este trigger garantiza que un evento no pueda programarse antes de la fecha de la actividad asociada.

  Activación: Se ejecuta **antes** de insertar o actualizar un registro en la tabla **EVENTO**.

  Acciones:

  - Compara la fecha y hora del evento (*fecha_hora*) con la fecha de la actividad asociada (*fecha_hora*).
  - Si *fecha_hora* del evento es anterior a *fecha_hora* de la actividad, cancela la operación y muestra un mensaje de error.

Nota: La capacidad de un trigger para responder a múltiples eventos depende del Sistema de Gestión de Bases de Datos (SGBD) utilizado.

- En algunos SGBD, como `MySQL`, un trigger solo puede estar asociado a un único evento (*INSERT*, *UPDATE* o *DELETE*) y a un único momento de ejecución (*BEFORE* o *AFTER*). Por lo tanto, si se requiere que un trigger maneje múltiples eventos o momentos, es necesario crear triggers separados para cada combinación.
- En otros SGBD, como `SQL Server` o `PostgreSQL`, es posible definir un solo trigger que responda a múltiples eventos y/o momentos. Esto permite manejar, por ejemplo, operaciones *INSERT* y *UPDATE* dentro del mismo trigger.

En nuestro caso concreto, estamos utilizando `MySQL` como gestor de bases de datos, y por lo tanto tendremos que crear cinco triggers diferentes para cubrir todas las combinaciones necesarias de eventos y momentos de ejecución.

3. Desarrollo de la base de datos en MySql

En este capítulo se implementará la base de datos siguiendo el modelo relacional definido en los capítulos anteriores. Se crearán todas las tablas especificadas, definiendo sus atributos y las restricciones necesarias para garantizar la integridad de los datos. Además, se programarán los triggers diseñados previamente para asegurar la consistencia de la información, automatizar procesos clave y mantener las reglas de negocio establecidas en el modelo. Todo esto se realizará utilizando SQL y MySQL como gestor de bases de datos.

3.1 Creación de la base de datos

```
    -- Creación de la base de datos
    CREATE DATABASE artevida_cultura;
    USE artevida_cultura;
```

3.2 Creación de tablas

```
-- Tabla ASISTENTE
CREATE TABLE ASISTENTE (
    id_asistente INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    telefono VARCHAR(20),
    email VARCHAR(100)
);

-- Tabla UBICACION
CREATE TABLE UBICACION (
    id_ubicacion INT AUTO_INCREMENT PRIMARY KEY,
    aforo INT NOT NULL CHECK (aforo >= 1),
    direccion VARCHAR(255),
    ciudad_pueblo VARCHAR(100),
    precio_alquiler DECIMAL(10,2),
    caracteristicas TEXT
);

-- Tabla ACTIVIDAD
CREATE TABLE ACTIVIDAD (
    id_actividad INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    direccion VARCHAR(255),
```

```
    fecha_hora DATETIME NOT NULL -- Cambiado de DATE a DATETIME para consistencia
);

-- Tabla ARTISTA
CREATE TABLE ARTISTA (
    id_artista INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    biografia TEXT
    -- Eliminado el atributo 'cache' de aquí
);

-- Tabla EVENTO
CREATE TABLE EVENTO (
    id_evento INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    descripcion TEXT,
    fecha_hora DATETIME NOT NULL,
    precio DECIMAL(10,2),
    id_ubicacion INT NOT NULL,
    id_actividad INT NOT NULL,
    estado_aforo BOOLEAN,
    FOREIGN KEY (id_ubicacion) REFERENCES UBICACION(id_ubicacion)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    FOREIGN KEY (id_actividad) REFERENCES ACTIVIDAD(id_actividad)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);

-- Tabla ASISTE (Relación N-M entre ASISTENTE y EVENTO)
CREATE TABLE ASISTE (
    id_evento INT,
    id_asistente INT,
    PRIMARY KEY (id_evento, id_asistente),
    FOREIGN KEY (id_evento) REFERENCES EVENTO(id_evento)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    FOREIGN KEY (id_asistente) REFERENCES ASISTENTE(id_asistente)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

-- Tabla ACTUA (Relación N-M entre ACTIVIDAD y ARTISTA)
CREATE TABLE ACTUA (
    id_actividad INT,
    id_artista INT,
    cache DECIMAL(10,2) NOT NULL CHECK (cache > 0),
    PRIMARY KEY (id_actividad, id_artista),
```

```
    FOREIGN KEY (id_actividad) REFERENCES ACTIVIDAD(id_actividad)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    FOREIGN KEY (id_artista) REFERENCES ARTISTA(id_artista)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

3.3 Creación de los Triggers

Nota: Para algunos triggers hemos usado expresiones regulares.

```
[breaklines=true]
-- Trigger 1: Validación del Aforo en UBICACION antes de insertar en ASISTE
DELIMITER $$

CREATE TRIGGER before_asiste_insert
BEFORE INSERT ON ASISTE
FOR EACH ROW
BEGIN
    DECLARE total_asistentes INT;
    DECLARE max_aforo INT;

    -- Obtener el aforo máximo y el número actual de asistentes en una sola consulta
    SELECT U.aforo, COUNT(A.id_asistente) INTO max_aforo, total_asistentes
    FROM UBICACION U
    JOIN EVENTO E ON U.id_ubicacion = E.id_ubicacion
    LEFT JOIN ASISTE A ON E.id_evento = A.id_evento
    WHERE E.id_evento = NEW.id_evento
    GROUP BY U.aforo;

    -- Verificar si el aforo se excedería con la nueva inserción
    IF (total_asistentes + 1) > max_aforo THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Aforo completo. No se puede agregar más asistentes.';
    ELSE
        -- Actualizar el estado_aforo en el evento
        IF (total_asistentes + 1) = max_aforo THEN
            UPDATE EVENTO
            SET estado_aforo = 1
            WHERE id_evento = NEW.id_evento;
        ELSE
            UPDATE EVENTO
            SET estado_aforo = 0
```

```
            WHERE id_evento = NEW.id_evento;
        END IF;
    END IF;
END$$

-- Trigger 2: Validación del Aforo en UBICACION antes de eliminar de ASISTE
CREATE TRIGGER before_asiste_delete
BEFORE DELETE ON ASISTE
FOR EACH ROW
BEGIN
    DECLARE total_asistentes INT;
    DECLARE max_aforo INT;

    -- Obtener el aforo máximo y el número actual de asistentes en una sola consulta
    SELECT U.aforo, COUNT(A.id_asistente) INTO max_aforo, total_asistentes
    FROM UBICACION U
    JOIN EVENTO E ON U.id_ubicacion = E.id_ubicacion
    LEFT JOIN ASISTE A ON E.id_evento = A.id_evento
    WHERE E.id_evento = OLD.id_evento
    GROUP BY U.aforo;

    -- Actualizar el estado_aforo en el evento
    IF total_asistentes < max_aforo THEN
        UPDATE EVENTO
        SET estado_aforo = 0
        WHERE id_evento = OLD.id_evento;
    ELSEIF total_asistentes = max_aforo THEN
        UPDATE EVENTO
        SET estado_aforo = 1
        WHERE id_evento = OLD.id_evento;
    END IF;
END$$

-- Trigger 3: Validación del Formato de Correo
-- y Teléfono en ASISTENTE antes de insertar


CREATE TRIGGER before_asistente_insert
BEFORE INSERT ON ASISTENTE
FOR EACH ROW
BEGIN
    -- Validación del formato de email utilizando expresión regular
    IF NEW.email IS NOT NULL AND NEW.email NOT REGEXP '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Formato de email inválido.';
    END IF;

    -- Validación de que 'telefono' contenga solo números
```

```
    IF NEW.telefono IS NOT NULL AND NEW.telefono NOT REGEXP '^[0-9]+$' THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'El teléfono debe contener solo números.';
    END IF;

    -- Validación de la longitud de 'telefono' (por ejemplo, entre 7 y 15 dígitos)
    IF NEW.telefono IS NOT NULL AND (CHAR_LENGTH(NEW.telefono) < 7 OR CHAR_LENGTH(NEW.telefono) > 15) T
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'El teléfono debe tener entre 7 y 15 dígitos.';
    END IF;
END$$



-- Trigger 4: Validación de Fechas en EVENTO antes de insertar
CREATE TRIGGER before_evento_insert
BEFORE INSERT ON EVENTO
FOR EACH ROW
BEGIN
    DECLARE fecha_actividad DATETIME;

    -- Obtener la fecha y hora de la actividad asociada
    SELECT fecha_hora INTO fecha_actividad
    FROM ACTIVIDAD
    WHERE id_actividad = NEW.id_actividad;

    -- Verificar que la fecha y hora del evento no sea anterior a la
    -- fecha y hora de la actividad

    IF NEW.fecha_hora < fecha_actividad THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'La fecha y hora del evento no puede ser anterior a la fecha y hora de l
    END IF;
END$$

-- Trigger 5: Validación de Fechas en EVENTO antes de actualizar
CREATE TRIGGER before_evento_update
BEFORE UPDATE ON EVENTO
FOR EACH ROW
BEGIN
    DECLARE fecha_actividad DATETIME;

    -- Obtener la fecha y hora de la actividad asociada
    SELECT fecha_hora INTO fecha_actividad
    FROM ACTIVIDAD
    WHERE id_actividad = NEW.id_actividad;
```

```
    -- Verificar que la fecha y hora del evento
    -- no sea anterior a la fecha y hora de la actividad

    IF NEW.fecha_hora < fecha_actividad THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'La fecha y hora del evento no puede ser anterior a la fecha y hora de l
    END IF;
END$$

DELIMITER ;
```

Nota: El uso de la instrucción `SIGNAL SQLSTATE '45000'` permite generar excepciones personalizadas en MySQL, lo cual es útil para implementar reglas de negocio directamente en la base de datos. A continuación, se presenta un resumen de sus componentes clave:

- **SIGNAL:** Inicia la generación de una excepción personalizada.
- **SQLSTATE '45000':** Código estándar para excepciones definidas por el usuario, utilizado cuando no existe un código de error específico.
- **SET MESSAGE_TEXT:** Define un mensaje descriptivo que explica la causa de la excepción.

4. Consultas y vistas

En este último capítulo insertaremos datos en nuestra base de datos recién creada, haremos una serie de consultas y crearemos un par de vistas interesantes.

4.1 Inserción de los datos en las tablas

Los datos que vamos a insertar en las tablas proceden de archivos `.csv`.

Nota importante: Cuando trabajamos con la carga de datos en MySQL utilizando el comando `LOAD DATA LOCAL INFILE`, estamos permitiendo que el servidor MySQL acceda a archivos locales del cliente (tu máquina). Por razones de seguridad, esta funcionalidad está deshabilitada de forma predeterminada en muchas configuraciones de MySQL, ya que podría ser explotada para cargar archivos no autorizados.

Por este motivo, es necesario realizar un cambio en la configuración del servidor para habilitar la opción `local_infile`, lo que permitirá que MySQL acepte la carga de datos desde archivos locales.

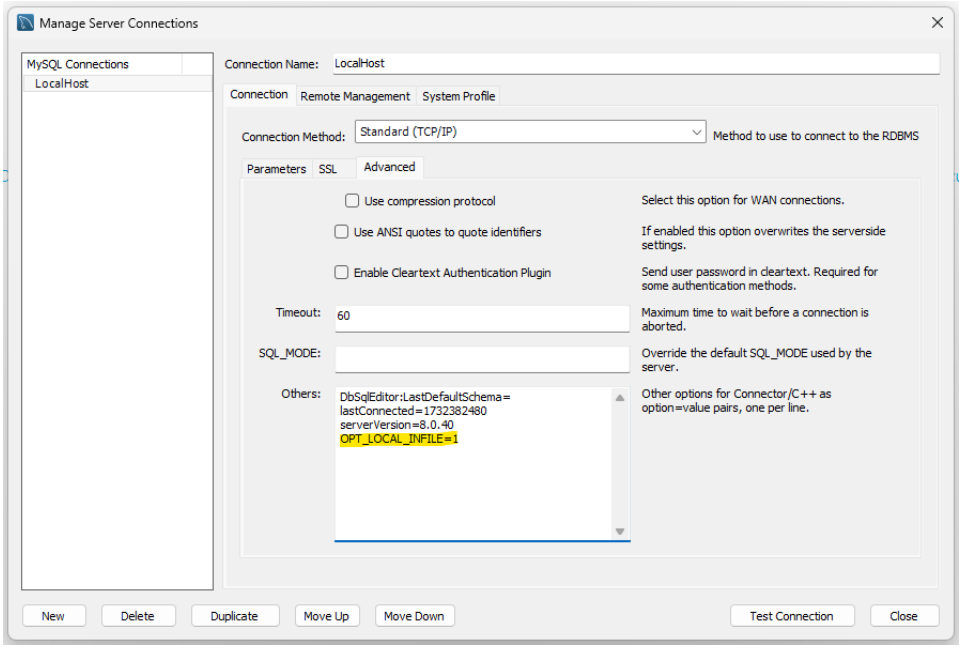


Figura 4.1: Configuración para habilitar local_infile en MySQL

Nota: Para el uso de los archivos .csv es necesario indicar la ruta completa.

Código de inserción

```
-- Cargar datos en la tabla ASISTENTE
LOAD DATA LOCAL INFILE 'D:/Master/Primer_Cuatri/SQL project/DataBase/data/ASISTENTE_Table.csv'
INTO TABLE asistente
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(nombre, telefono, email);

-- ------------------ Insercción de datos desde arhcivos .csv --------------


-- Cargar datos en la tabla UBICACION
LOAD DATA LOCAL INFILE 'D:/Master/Primer_Cuatri/SQL project/DataBase/data/UBICACION_Table.csv'
INTO TABLE UBICACION
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(id_ubicacion, aforo, direccion, ciudad_pueblo, precio_alquiler, caracteristicas);

-- Cargar datos en la tabla ACTIVIDAD
LOAD DATA LOCAL INFILE 'D:/Master/Primer_Cuatri/SQL project/DataBase/data/ACTIVIDAD_Table.csv'
INTO TABLE ACTIVIDAD
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(id_actividad, nombre, direccion, fecha_hora);

-- Cargar datos en la tabla ARTISTA
LOAD DATA LOCAL INFILE 'D:/Master/Primer_Cuatri/SQL project/DataBase/data/ARTISTA_Table.csv'
INTO TABLE ARTISTA
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS
(id_artista, nombre, biografia);

-- Cargar datos en la tabla EVENTO
LOAD DATA LOCAL INFILE 'D:/Master/Primer_Cuatri/SQL project/DataBase/data/EVENTO_Table.csv'
INTO TABLE EVENTO
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

```
IGNORE 1 ROWS
(id_evento, nombre, descripcion, fecha_hora, precio, id_ubicacion, id_actividad, estado_aforo);

-- Cargar datos en la tabla ASISTE
LOAD DATA LOCAL INFILE 'D:/Master/Primer_Cuatri/SQL project/DataBase/data/ASISTE_Table.csv'
INTO TABLE ASISTE
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(id_evento, id_asistente);

-- Cargar datos en la tabla ACTUA
LOAD DATA LOCAL INFILE 'D:/Master/Primer_Cuatri/SQL project/DataBase/data/ACTUA_Table.csv'
INTO TABLE ACTUA
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS
(id_actividad, id_artista, cache);
```

4.2 Consultas SQL con Explicaciones

4.2.1 Consulta de eventos con asistentes al límite del aforo

Descripción: Esta consulta devuelve los eventos en los que el número de asistentes alcanzó exactamente el aforo máximo permitido en la ubicación asignada.

```
SELECT
    evento.nombre AS evento,
    COUNT(asiste.id_asistente) AS total_asistentes,
    ubicacion.aforo AS aforo_maximo
FROM evento
JOIN asiste ON evento.id_evento = asiste.id_evento
JOIN ubicacion ON evento.id_ubicacion = ubicacion.id_ubicacion
GROUP BY evento.id_evento, ubicacion.aforo
HAVING COUNT(asiste.id_asistente) = ubicacion.aforo;
```

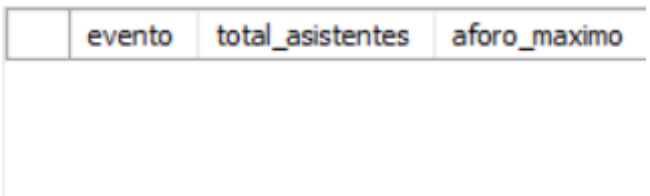


Figura 4.2: Eventos con asistentes al límite del aforo

4.2.2 Actividades con artistas y sus caches totales

Descripción: Lista las actividades, los artistas que participaron en ellas y la suma total del caché pagado a cada artista por actividad.

```
SELECT
    actividad.nombre AS actividad,
    artista.nombre AS artista,
    SUM(actua.cache) AS cache_total
FROM actividad
JOIN actua ON actividad.id_actividad = actua.id_actividad
JOIN artista ON actua.id_artista = artista.id_artista
GROUP BY actividad.id_actividad, artista.id_artista
ORDER BY cache_total DESC;
```

actividad	artista	cache_total
Barrister	Dua Lipa	491963.41
Clinical molecular geneticist	Beyoncé	484610.38
Call centre manager	Billie Eilish	477078.44
Clinical molecular geneticist	Ed Sheeran	467194.32
Audiological scientist	Ed Sheeran	456981.05

Figura 4.3: Actividades y caches totales por artista

4.2.3 Total de asistentes por evento y su ubicación

Descripción: Calcula el total de asistentes a cada evento junto con la ubicación asociada.

```
SELECT
    ubicacion.id_ubicacion AS id_ubicacion,
    evento.nombre AS evento,
    COUNT(asiste.id_asistente) AS total_asistentes
FROM ubicacion
JOIN evento ON ubicacion.id_ubicacion = evento.id_ubicacion
JOIN asiste ON evento.id_evento = asiste.id_evento
GROUP BY ubicacion.id_ubicacion, evento.id_evento;
```

id_ubicacion	evento	total_asistentes
10	Transform magnetic architectures	1
11	Visualize proactive applications	5
17	Optimize visionary metrics	1
4	Incubate proactive content	5
15	Orchestrate seamless roi	4

Figura 4.4: Total de asistentes por evento y su ubicación

4.2.4 Eventos con ingresos totales por entradas

Descripción: Calcula los ingresos generados por cada evento multiplicando el precio de la entrada por la cantidad de asistentes.

```
SELECT
    evento.nombre AS evento,
    COUNT(asiste.id_asistente) * evento.precio AS ingresos_totales
FROM evento
JOIN asiste ON evento.id_evento = asiste.id_evento
GROUP BY evento.id_evento
ORDER BY ingresos_totales DESC;
```

evento	ingresos_totales
Grow frictionless channels	1529.57
Redefine dot-com metrics	1491.55
Productize global web services	1462.25
Incubate proactive content	1432.30
Unleash best-of-breed e-tailers	1370.65

Figura 4.5: Ingresos totales por evento

4.2.5 Promedio del caché por artista

Descripción: Devuelve el promedio del caché pagado a cada artista considerando todas las actividades en las que participaron.

```
SELECT
    artista.nombre AS artista,
    AVG(actua.cache) AS promedio_cache
FROM artista
JOIN actua ON artista.id_artista = actua.id_artista
GROUP BY artista.id_artista
ORDER BY promedio_cache DESC;
```

	artista	promedio_cache
▶	Ed Sheeran	402339.607500
	Shawn Mendes	298846.237143
	Beyoncé	294092.290000
	Billie Eilish	259572.895000
	Dua Lipa	236198.090000

Figura 4.6: Promedio del caché por artista

4.2.6 Eventos sin asistentes

Descripción: Encuentra los eventos que no tienen asistentes registrados en la tabla `asiste`.

```
SELECT
    evento.nombre AS evento,
    evento.fecha_hora AS fecha_evento
FROM evento
LEFT JOIN asiste ON evento.id_evento = asiste.id_evento
WHERE asiste.id_asistente IS NULL;
```

	evento	fecha_evento
▶	Utilize transparent infrastructures	2024-09-12 20:40:44

Figura 4.7: Eventos sin asistentes

4.2.7 Artistas con más de 3 actividades

Descripción: Devuelve los nombres de los artistas que han participado en más de 3 actividades distintas.

```
SELECT
    artista.nombre AS artista,
    COUNT(DISTINCT actua.id_actividad) AS total_actividades
FROM artista
JOIN actua ON artista.id_artista = actua.id_artista
GROUP BY artista.id_artista
HAVING COUNT(DISTINCT actua.id_actividad) > 3;
```

	artista	total_actividades
▶	Beyoncé	5
	Taylor Swift	4
	Ed Sheeran	4
	Drake	5
	Billie Eilish	6

Figura 4.8: Artistas con más de 3 actividades

4.2.8 Actividades y el total de asistentes a sus eventos relacionados

Descripción: Calcula el número total de asistentes a todos los eventos relacionados con cada actividad.

```
SELECT
    actividad.nombre AS actividad,
    COUNT(asiste.id_asistente) AS total_asistentes
FROM actividad
JOIN evento ON actividad.id_actividad = evento.id_actividad
JOIN asiste ON evento.id_evento = asiste.id_evento
GROUP BY actividad.id_actividad
ORDER BY total_asistentes DESC;
```

	actividad	total_asistentes
▶	Dramatherapist	19
	Ship broker	17
	Magazine features editor	16
	Horticultural consultant	14
	Trading standards officer	12

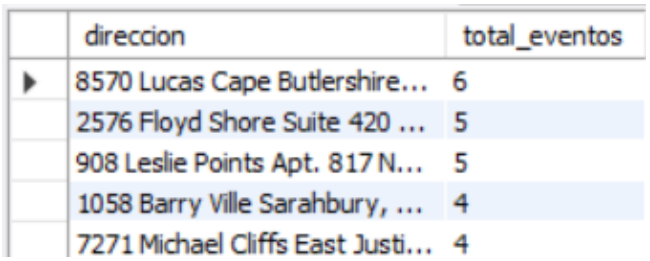
Figura 4.9: Total de asistentes por actividad relacionada

4.2.9 Ubicaciones con más eventos realizados

Descripción: Lista las ubicaciones y el número de eventos realizados en cada una de ellas, ordenadas por el número de eventos en orden descendente.

```
SELECT
```

```
    ubicacion.direccion AS direccion,
    COUNT(evento.id_evento) AS total_eventos
FROM ubicacion
JOIN evento ON ubicacion.id_ubicacion = evento.id_ubicacion
GROUP BY ubicacion.direccion
ORDER BY total_eventos DESC;
```



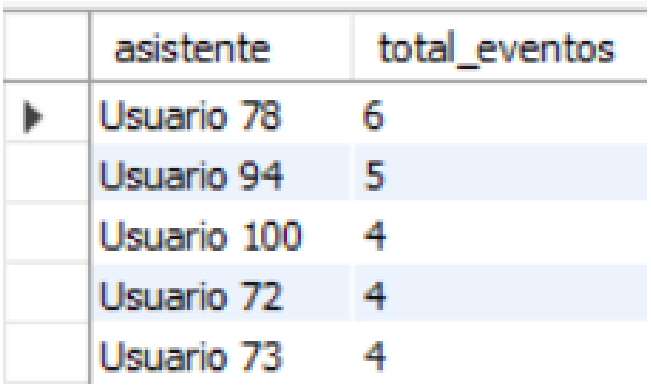
direccion	total_eventos
8570 Lucas Cape Butlershire...	6
2576 Floyd Shore Suite 420 ...	5
908 Leslie Points Apt. 817 N...	5
1058 Barry Ville Sarahbury, ...	4
7271 Michael Cliffs East Justi...	4

Figura 4.10: Ubicaciones con más eventos realizados

4.2.10 Asistentes que han participado en más de un evento

Descripción: Devuelve los nombres de los asistentes que han asistido a más de un evento y el número total de eventos a los que han asistido.

```
SELECT
    asistente.nombre AS asistente,
    COUNT(DISTINCT asiste.id_evento) AS total_eventos
FROM asistente
JOIN asiste ON asistente.id_asistente = asiste.id_asistente
GROUP BY asistente.id_asistente
HAVING COUNT(DISTINCT asiste.id_evento) > 1
ORDER BY total_eventos DESC;
```



asistente	total_eventos
Usuario 78	6
Usuario 94	5
Usuario 100	4
Usuario 72	4
Usuario 73	4

Figura 4.11: Asistentes con participación en más de un evento

4.2.11 Vista: Balance Anual de los Eventos

Descripción: Esta vista, llamada `evento_balance_anual`, calcula las ganancias o pérdidas de cada evento en un año. Tiene en cuenta los ingresos generados por las entradas y los costos asociados, como el alquiler de la ubicación y los cachés de los artistas que participaron.

Elementos principales de la vista:

- **Año del evento:** Los eventos se agrupan por el año en el que se realizaron.
- **Costo de la ubicación:** Se considera el precio de alquiler de la ubicación asociada al evento.
- **Costo de los artistas:** Calcula la suma de los cachés de todos los artistas involucrados en la actividad relacionada con el evento.
- **Ingresos totales:** Multiplica el precio de las entradas del evento por el número de asistentes.
- **Balance:** Calcula el resultado final de ingresos menos costos. Si es negativo, representa una pérdida; si es positivo, una ganancia.

Código SQL de la vista:

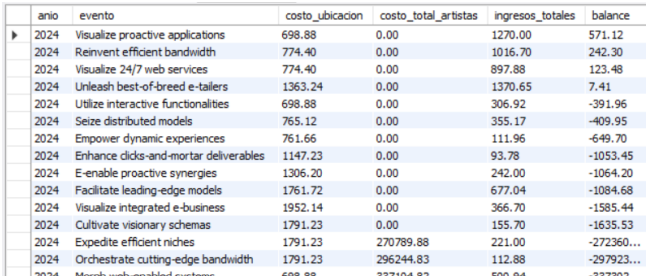
```
CREATE VIEW evento_balance_anual AS
SELECT
    YEAR(evento.fecha_hora) AS anio,
    evento.nombre AS evento,
    ubicacion.precio_alquiler AS costo_ubicacion,
    IFNULL(SUM(actua.cache), 0) AS costo_total_artistas,
    IFNULL(COUNT(asiste.id_asistente) * evento.precio, 0) AS ingresos_totales,
    (IFNULL(COUNT(asiste.id_asistente) * evento.precio, 0) -
     (ubicacion.precio_alquiler + IFNULL(SUM(actua.cache), 0))) AS balance
FROM evento
LEFT JOIN ubicacion ON evento.id_ubicacion = ubicacion.id_ubicacion
LEFT JOIN actua ON evento.id_actividad = actua.id_actividad
LEFT JOIN asiste ON evento.id_evento = asiste.id_evento
GROUP BY anio, evento.id_evento, ubicacion.precio_alquiler
ORDER BY anio, balance DESC;
```

Explicación del cálculo:

- **Ingresos Totales:** Calculados multiplicando el precio de las entradas del evento por el número de asistentes registrados.
- **Costo Total:** Suma del precio de alquiler de la ubicación y los cachés de los artistas involucrados.
- **Balance:** Ingresos totales menos costos totales.
- Si no hay asistentes o artistas asociados, se utiliza `IFNULL` para tratar estos valores como cero.

Consulta a la vista:

```
SELECT * FROM evento_balance_anual;
```



anio	evento	costo_ubicacion	costo_total_artistas	ingresos_totales	balance
2024	Visualize proactive applications	698.88	0.00	1270.00	571.12
2024	Reinvent efficient bandwidth	774.40	0.00	1016.70	242.30
2024	Visualize 24/7 web services	774.40	0.00	897.88	123.48
2024	Unleash best-of-breed e-tailers	1363.24	0.00	1370.65	7.41
2024	Utilize interactive functionalities	698.88	0.00	306.92	-391.96
2024	Seize distributed models	765.12	0.00	355.17	-409.95
2024	Empower dynamic experiences	761.66	0.00	111.96	-649.70
2024	Enhance clicks-and-mortar deliverables	1147.23	0.00	93.78	-1053.45
2024	E-enable proactive synergies	1306.20	0.00	242.00	-1064.20
2024	Facilitate leading-edge models	1761.72	0.00	677.04	-1084.68
2024	Visualize integrated e-business	1952.14	0.00	366.70	-1585.44
2024	Cultivate visionary schemas	1791.23	0.00	155.70	-1635.53
2024	Expedite efficient niches	1791.23	270789.88	221.00	-272360...
2024	Orchestrate cutting-edge bandwidth	1791.23	296244.83	112.88	-297923...

Figura 4.12: Representación visual de los balances de los eventos por año.