Proyecto Final SQL: Creación de una Base de Datos

Máster Big Data, Data Science & Inteligencia Artificial

Proyecto realizado por: Alejandro Borrego Megías

Correo: alejbormeg@gmail.com

Fecha: 17 de Noviembre de 2023

Índice

Resumen

El presente documento detalla el proyecto final realizado como parte del Máster en Big Data, Data Science & Inteligencia Artificial. El proyecto se enfoca en la creación de una base de datos utilizando SQL y se abordarán todas las fases del proceso desde la obtención de los requerimientos hasta la creación en la base de datos **MySQL**.

Objetivos del Proyecto

1. **Diseño del modelo Entidad-Relación:** En esta etapa, el objetivo principal es conceptualizar la estructura de la base de datos mediante el diseño de un modelo Entidad-Relación (ER). Este modelo visual representa las entidades del sistema, sus atributos y las relaciones entre ellas.

2. **Paso a tablas:** Una vez completado el diseño ER, el siguiente paso es la transformación de este modelo conceptual en un conjunto de tablas. Este proceso implica definir las tablas, sus campos y las restricciones de integridad referencial.

3. **Creación de la base de datos:** se procede a la creación física de la base de datos con **MySQL**. Este paso implica la ejecución de scripts SQL para establecer la estructura de la base de datos, incluyendo la creación de tablas, definición de claves primarias y foráneas, y la inserción de datos entre otros.

4. **Implementación de Triggers y vistas:** Para mejorar la funcionalidad y el control de la base de datos, se implementan *triggers* y *vistas*. Los *triggers* son procedimientos almacenados que se ejecutan automáticamente en respuesta a ciertos eventos, mientras que las *vistas* ofrecen perspectivas predefinidas de los datos.

5. **Consultas:** Finalmente, se definen y ejecutan consultas SQL para extraer información específica de la base de datos. En total se realizarán unas 13 consultas de diversa dificultad.

Enunciado

El enunciado de la actividad es el siguiente:

"Tenemos una empresa dedicaba a la organización de eventos culturales únicos “ArteVida Cultural”. Organizamos desde deslumbrantes conciertos de música clásica hasta exposiciones de arte vanguardista, pasando por apasionantes obras de teatro y cautivadoras conferencias, llevamos la cultura a todos los rincones de la comunidad.

Necesitamos gestionar la gran variedad de eventos y detalles, así como las ganancias que obtenemos. Para ello, es necesario llevar un registro adecuado de cada evento, de los artistas que los protagonizan, las ubicaciones donde tienen lugar, la venta de entradas y, por supuesto, el entusiasmo de los visitantes que asisten.

Hemos decidido diseñar e implementar una base de datos relacional que no solo simplifique la organización de eventos, sino que también permita analizar datos valiosos para tomar decisiones informadas.

En nuestra empresa ofrecemos una serie de actividades que tienen un nombre, un tipo: concierto de distintos tipos de música (clásica, pop, blues, soul, rock and roll, jazz, reggaeton, góspel, country, ...), exposiciones, obras de teatro y conferencias, aunque en un futuro estamos dispuestos a organizar otras actividades. Además, en cada actividad participa uno o varios artistas y un coste (suma del caché de los artistas).

El artista tiene un nombre, un caché que depende de la actividad en la que participe y una breve biografía.

La ubicación tendrá un nombre (Teatro Maria Guerrero, Estadio Santiago Bernabeu,...), dirección, ciudad o pueblo, aforo, precio del alquiler y características.

De cada evento tenemos que saber el nombre del evento (p.e. “VI festival de música clásica de Alcobendas”), la actividad, la ubicación, el precio de la entrada, la fecha y la hora, así como una breve descripción del mismo. En un evento sólo se realiza una actividad.

También tendremos en cuenta los asistentes a los eventos, de los que sabemos su nombre completo, sus teléfonos de contacto y su email. Una persona puede asistir a más de un evento y a un evento pueden asistir varias personas. Nos interesará realizar consultas por el tipo de actividad, en que fecha se han realizado más eventos, en qué ciudad realizamos más eventos"

Diseño del modelo Entidad-Relación

Entidades

En base al enunciado, las entidades propuestas son las siguientes:

- **Artista**: En esta entidad almacenamos toda la información relativa a los artistas (biografía y nombre, tomando el nombre como clave primaria). En lo relativo al *Caché* que se menciona en el enunciado, dado que es dependiente de la actividad en la que participe el Artista, hemos decidido ponerlo como atributo de la relación entre Artista y Actividad.
- **Actividad**: Esta entidad encapsula todo lo relativo a las actividades que la empresa organiza: el tipo de actividad, nombre y además añadimos un atributo extra *CodActividad* que será la clave primaria. Añadimos esta propiedad extra porque analizando los demás atributos no pensamos que el *nombre* de la actividad baste como clave primaria, pudiendo añadirse quizá actividades con mismo nombre pero

distinta tipología. Por otra parte, se va a limitar el valor del atributo tipo a los que se comentan en el enunciado (concierto, exposiciones, obras de teatro y conferencias) añadiendo un tipo extra *actividad genérica* en caso de que la actividad que se inserta no encaja en ninguno de los anteriores. En un futuro si se quiere añadir nuevos tipos basta con hacer una migración a la base de datos añadiendo más valores posibles. Por otro lado, para abordar el asunto de los subtipos (en el enunciado solo se mencionan subtipos de conciertos) vamos a crear una entidad nueva que presentamos a continuación. Para el requerimiento del Caché total de cada actividad (suma de los cachés de los artistas que en ella participan) nos beneficiaremos de la relación que une cada actividad con los artistas, de manera que aunque no se guarde explícitamente, mediante una consulta podría obtenerse con facilidad.

- **Subtipo**: Esta actividad solo tiene dos atributos (CodSubtipo que será la clave primaria y Subtipo). Esta entidad se relaciona con las actividades que incluyan subtipos, de forma que además no permite manejar con flexibilidad la posibilidad de que haya actividades con varios subtipos simultáneamente y hacer consultas relativas a ellos con facilidad. Además nos aporta mayor flexibilidad pues no solo se añaden subtipos para conciertos, sino para cualquier otra categoría si fuera necesario.
- **Ubicación**: Representa una localización dónde se puede celebrar un Evento, y sus atributos son: CodUbicación (clave primaria), Nombre, Dirección, Ciudad, Aforo, Precio Alquiler y Características.
- **Asistente**: Representa una persona que asiste a un evento, y de ella almacenamos su nombre, Teléfono y Email (clave primaria).
- **Evento**: Se trata de la entidad principal, con ella se relacionan de forma directa o indirecta todas las anteriores y tiene los siguientes atributos: descripción, hora, fecha, Precio de entrada, CodUbicación, NombreActividad, Nombre Evento y CodEvento. La clave primaria será CodEvento.

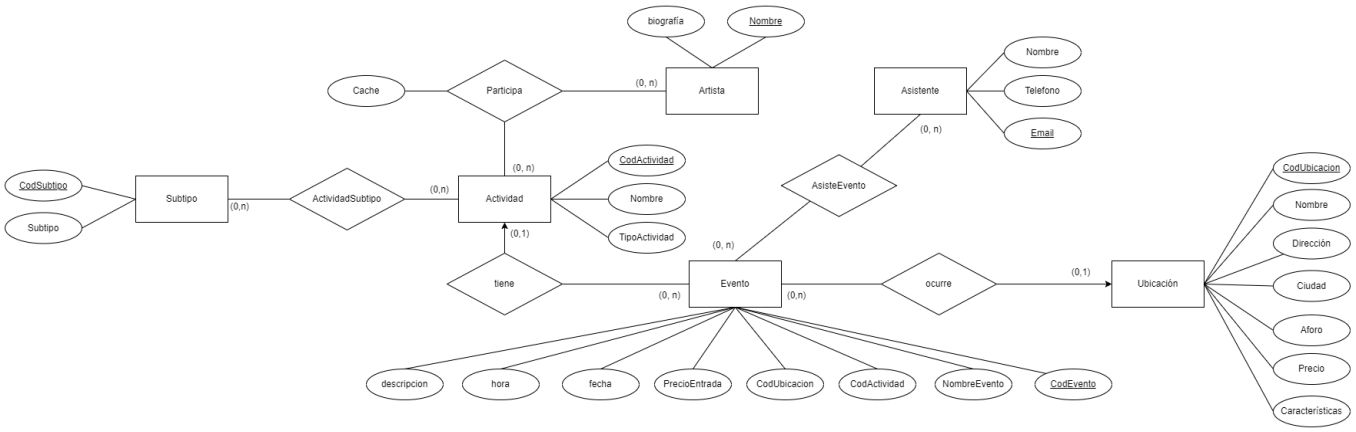
Relaciones

- **Participa**: Relaciona los artistas con las actividades. Se trata de una relación muchos a muchos. Por otro lado, las cardinalidades son (0, n) en ambos sentidos pues se permite que haya Actividades de las que aún no se conocen los artistas que participan y Artistas que no han participado en ninguna Actividad aún. En el paso a tablas, esta relación se transforma en una tabla.
- **ActividadSubtipo**: Relaciona las Actividades con los Subtipos y es de muchos a muchos. De nuevo, la cardinalidad en ambos sentidos es (0, n) pues se permite que existan Subtipos sin Actividad asociada y Actividades sin Subtipos asociados.
- **Tiene**: Relación uno a muchos entre varias Actividades y un Evento (pues en el enunciado se especifica que en un evento sólo ocurre una actividad). Esta relación, en el paso a tablas, esta relación se transforma en añadir a la tabla Evento el código de actividad que ocurre que se trata de una clave externa. La cardinalidad es de (1,1) por el lado de las Actividades y de (0, n) por el lado de Evento (esto permite que haya eventos sin actividad asociada en la tabla).
- **AsisteEvento**: Relación muchos a muchos entre Evento y sus Asistentes. De nuevo las cardinalidades (0,n) por ambas partes permitiendo que haya asistentes registrados sin eventos a los que ir de momento o eventos sin asistentes. De esta relación nace una tabla de nuestra base de datos.
- **ocurre**: Relación uno a muchos entre varios Eventos y una única Ubicación. Esto, en el paso a tablas pasa a añadirse una clave externa en la tabla Evento con el Código de Ubicación.

Nota: Se ha optado por la cardinalidad (0,n) en todos los casos porque permite flexibilidad a la hora de insertar tuplas en las tablas y no había restricciones en el enunciado al respecto. Así, se entiende que puede existir un evento del que todavía no se conoce Ubicación ni Actividad porque se está pensando aún, o se permite tener asistentes registrados en el sistema porque se han hecho una cuenta en la plataforma de la empresa aunque aún no hayan asistido ni vayan a asisitir a algún evento.

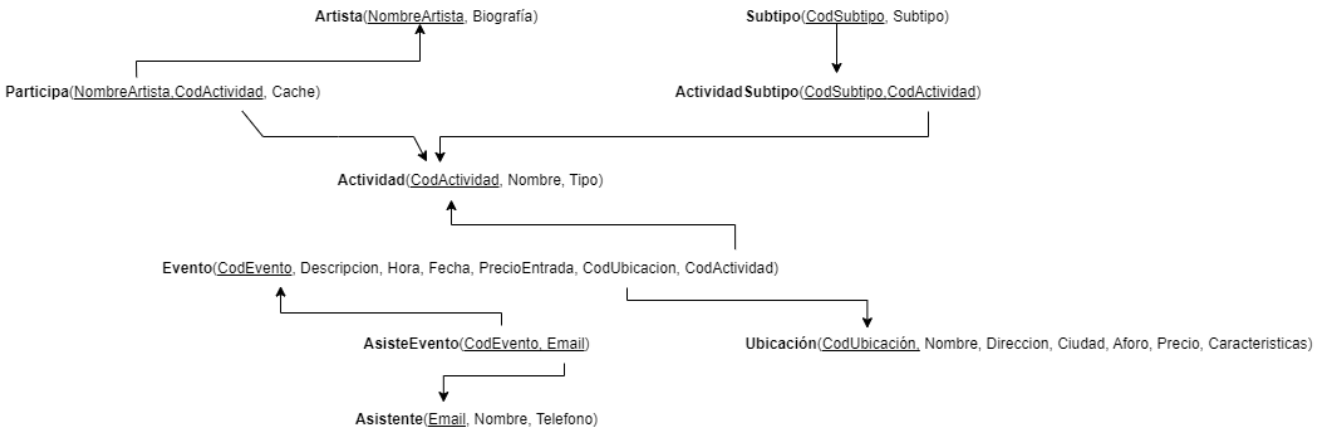
Diagrama E-R

Según todo lo anterior, el diagrama Entidad-Relación propuesto es el siguiente:



Paso a tablas

En base al diagrama obtenido en la anterior fase, el correspondiente paso a tablas sería el siguiente:



Formas normales

Script SQL

```
/*
Autor: Alejandro Borrego Megías

Creación de la base de datos, las tablas e inserción de datos
*/

DROP DATABASE IF EXISTS ProyectoMySQL;

CREATE DATABASE ProyectoMySQL;

USE ProyectoMySQL;

/* ----------------------------------------------------------------------------
-----------------
```

```
Definición de la estructura de la base de datos
-------------------------------------------------------------------------------
---------------*/
-- Creación de la tabla Artista
CREATE TABLE IF NOT EXISTS Artista (
    NombreArtista VARCHAR(50) PRIMARY KEY,
    Biografia VARCHAR(300)
);

-- Creación de la tabla Asistente
CREATE TABLE IF NOT EXISTS Asistente (
    Email VARCHAR(50) PRIMARY KEY,
    Nombre VARCHAR(50),
    Telefono VARCHAR(20)
);

-- Creación de la tabla Ubicación
CREATE TABLE IF NOT EXISTS Ubicacion (
    CodUbicacion INT AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(50),
    Direccion VARCHAR(100),
    Ciudad VARCHAR(50),
    Aforo INT,
    Precio DECIMAL(10, 2),
    Caracteristicas VARCHAR(200)
);

CREATE TABLE IF NOT EXISTS Actividad (
    CodActividad INT AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(255),
    Tipo ENUM('concierto', 'exposiciones', 'obra_de_teatro', 'conferencia',
'actividad_generica')
);

-- Creación de la tabla Evento
CREATE TABLE IF NOT EXISTS Evento (
    CodEvento INT AUTO_INCREMENT PRIMARY KEY,
    NombreEvento VARCHAR(200),
    Descripcion VARCHAR(200),
    Hora TIME,
    Fecha DATE,
    PrecioEntrada DECIMAL(10, 2),
    CodUbicacion INT,
    CodActividad INT,
    FOREIGN KEY (CodUbicacion) REFERENCES Ubicacion(CodUbicacion) ON DELETE NO
ACTION,
    FOREIGN KEY (CodActividad) REFERENCES Actividad(CodActividad) ON DELETE NO
ACTION
);

-- Creación de la tabla AsisteEvento
CREATE TABLE IF NOT EXISTS AsisteEvento (
    CodEvento INT,
    Email VARCHAR(50),
```

```
    PRIMARY KEY (CodEvento, Email),
    FOREIGN KEY (CodEvento) REFERENCES Evento(CodEvento) ON DELETE CASCADE,
    FOREIGN KEY (Email) REFERENCES Asistente(Email) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS Participa (
    NombreArtista VARCHAR(255),
    CodActividad INT,
    Cache DECIMAL(10, 2),
    PRIMARY KEY (NombreArtista, CodActividad),
    FOREIGN KEY (NombreArtista) REFERENCES Artista(NombreArtista) ON DELETE
CASCADE,
    FOREIGN KEY (CodActividad) REFERENCES Actividad(CodActividad) ON DELETE
CASCADE
);

CREATE TABLE IF NOT EXISTS Subtipo (
    CodSubtipo INT AUTO_INCREMENT PRIMARY KEY,
    Subtipo VARCHAR(255)
);

CREATE TABLE IF NOT EXISTS ActividadSubtipo (
    CodSubtipo INT,
    CodActividad INT,
    PRIMARY KEY (CodSubtipo, CodActividad),
    FOREIGN KEY (CodSubtipo) REFERENCES Subtipo(CodSubtipo) ON DELETE CASCADE,
    FOREIGN KEY (CodActividad) REFERENCES Actividad(CodActividad) ON DELETE
CASCADE
);

/*--------------------------------------------------------------------------------
----------------------
Trigger
Inserción de datos
--------------------------------------------------------------------------------
---------------------*/

INSERT INTO Artista (NombreArtista, Biografia) VALUES
    ('Melendi', 'Artista español que lleva cantando muchos años'),
    ('Morat', 'Grupo Colombiano de música pop-country'),
    ('JaimeLlorente', 'Actor Español'),
    ('Plácido Domingo', 'Cantante de Ópera español'),
    ('Pablo Alborán', 'Cantautor y músico español reconocido internacionalmente'),
    ('Blanca Suárez', 'Actriz española de cine y televisión');

INSERT INTO Asistente (Email, Nombre, Telefono) VALUES
    ('juanPer34@example.com', 'Juan Pérez', '+1234567890'),
    ('M.gonzalez_99@example.com', 'María González', '+9876543210'),
    ('luisRodri98@example.com', 'Luis Rodríguez', '+1122334455'),
    ('anaNavarro@example.com', 'Ana Navarro', '+1122334455'),
    ('davidGomez@example.com', 'David Gómez', '+9876543210'),
    ('mariaLopez@example.com', 'María López', '+1234567890');

INSERT INTO Ubicacion (Nombre, Direccion, Ciudad, Aforo, Precio, Caracteristicas)
```

```
VALUES
    ('Palacio de deportes', 'Calle A, 123', 'Granada', 1, 75.00, 'Características
de palacio de deportes'),
    ('Wizink center', 'Avenida B, 456', 'Madrid', 1500, 40.00, 'Características de
wizink center'),
    ('Teatro Cervantes', 'Calle C, 789', 'Málaga', 800, 60.00, 'Características de
Teatro Cervantes'),
    ('Teatro Real', 'Plaza de Oriente, 5', 'Madrid', 500, 90.00, 'Teatro lírico
español'),
    ('Alhambra', 'Calle Real, s/n', 'Granada', 300, 120.00, 'Palacio y fortaleza
en la ciudad de Granada'),
    ('Guggenheim Bilbao', 'Abandoibarra Etorb., 2', 'Bilbao', 1000, 50.00, 'Museo
de arte contemporáneo');


INSERT INTO Actividad (Nombre, Tipo) VALUES
    ('Concierto en Vivo', 'concierto'),
    ('Exposición de Arte Moderno', 'exposiciones'),
    ('Obra de Teatro Clásica', 'obra_de_teatro'),
    ('Drama Romántico', 'obra_de_teatro'),
    ('Flamenco en Vivo', 'concierto'),
    ('Cine Español Contemporáneo', 'exposiciones'),
    ('Monólogo de Blanca Suárez', 'obra_de_teatro');

INSERT INTO Evento (NombreEvento, Descripcion, Hora, Fecha, PrecioEntrada,
CodUbicacion, CodActividad) VALUES
    ('Madrid Sound', 'Concierto en Vivo en Madrid', '19:00:00', '2023-12-15',
30.00, 2, 1),
    ('Festival de Musica y Arte de Granada', 'Exposición de Arte Moderno',
'14:00:00', '2023-11-20', 10.00, 1, 2),
    ('Semana de Calderón', 'Obra de Teatro Clásica', '20:30:00', '2023-12-05',
25.00, 3, 3),
    ('Romeo y Julieta', 'Drama Romántico', '21:30:00', '2023-12-08', 97.00, 3, 4),
    ('Noche de Ópera en el Teatro Real', 'Ópera en Vivo en Madrid', '20:00:00',
'2023-12-10', 50.00, 4, 5),
    ('Concierto Flamenco en la Alhambra', 'Flamenco en Vivo en la Alhambra',
'21:00:00', '2023-11-25', 35.00, 2, 6),
    ('Exposición de Arte Contemporáneo en el Guggenheim Bilbao', 'Cine Español
Contemporáneo', '18:30:00', '2023-12-01', 15.00, 3, 7),
    ('Recital de Pablo Alborán', 'Concierto en Vivo en Madrid', '19:30:00', '2023-
11-30', 75.00, 1, 1);

INSERT INTO AsisteEvento (CodEvento, Email) VALUES
    (1, 'juanPer34@example.com'),
    (2, 'M.gonzalez_99@example.com'),
    (3, 'luisRodri98@example.com'),
    (5, 'anaNavarro@example.com'),
    (6, 'davidGomez@example.com'),
    (7, 'mariaLopez@example.com'),
    (5, 'davidGomez@example.com'),
    (5, 'mariaLopez@example.com');

INSERT INTO Participa (NombreArtista, CodActividad, Cache) VALUES
    ('Melendi', 1, 1000.00),
```

```
    ('Morat', 1, 800.00),
    ('Morat', 3, 900.00),
    ('JaimeLlorente', 2, 1200.00),
    ('Plácido Domingo', 5, 1200.00),
    ('Pablo Alborán', 1, 1500.00),
    ('Blanca Suárez', 6, 1000.00);

INSERT INTO Subtipo (Subtipo) VALUES
    ('Rock'),
    ('Modernismo'),
    ('Barroco'),
    ('Flamenco'),
    ('Cine Contemporáneo'),
    ('Monólogo'),
    ('Ópera');

INSERT INTO ActividadSubtipo (CodSubtipo, CodActividad) VALUES
    (1, 1),
    (2, 2),
    (3, 3),
    (7, 5),
    (5, 6),
    (6, 7);


-- TRIGGERS:
-- Creamos tabla para el trigger
CREATE TABLE IF NOT EXISTS ArtistaRegistroBiografias (
    ChangeID INT AUTO_INCREMENT PRIMARY KEY,
    NombreArtista VARCHAR(50),
    BiografiaOld VARCHAR(300),
    BiografiaNew VARCHAR(300),
    ChangeTimestamp TIMESTAMP
);

-- Creamos un trigger para insertar por cada vez que se cambie o actualice la
biografía de un cantante una tupla con la biografía anterior y la nueva
DELIMITER //
CREATE TRIGGER Artista_RegistroBiografias
AFTER UPDATE ON Artista
FOR EACH ROW
BEGIN
    INSERT INTO ArtistaRegistroBiografias (NombreArtista, BiografiaOld,
BiografiaNew, ChangeTimestamp)
    VALUES (OLD.NombreArtista, OLD.Biografia, NEW.Biografia, NOW());
END;
//
DELIMITER ;

-- Probamos el trigger
UPDATE Artista
SET Biografia = 'Ya no canta más, ahora es humorista'
WHERE NombreArtista = 'Melendi';
```

```
-- Comprobamos que se ha introducido la tupla correctamente
SELECT * FROM ArtistaRegistroBiografias;

-- Creamos un trigger para tener un histórico de los cambios de precios en las
ubicaciones
-- Primero creamos la tabla a rellenar por cada nuevo precio
CREATE TABLE IF NOT EXISTS UbicacionRegistroPrecios (
    ChangeID INT AUTO_INCREMENT PRIMARY KEY,
    CodUbicacion INT,
    PrecioOld DECIMAL(10, 2),
    PrecioNew DECIMAL(10, 2),
    ChangeTimestamp TIMESTAMP
);

-- Luego creamos el trigger
DELIMITER //
CREATE TRIGGER Ubicacion_CambioPrecios
AFTER UPDATE ON Ubicacion
FOR EACH ROW
BEGIN
    IF OLD.Precio <> NEW.Precio THEN
        INSERT INTO UbicacionRegistroPrecios (CodUbicacion, PrecioOld, PrecioNew,
ChangeTimestamp)
        VALUES (OLD.CodUbicacion, OLD.Precio, NEW.Precio, NOW());
    END IF;
END;
//
DELIMITER ;

-- Actualizamos una entrada con un precio nuevo
UPDATE Ubicacion
SET Precio = 50.00
WHERE CodUbicacion = 1;

-- Comprobamos el funcionamiento del trigger viendo cómo el precio pasa de 75 a 50
SELECT * FROM UbicacionRegistroPrecios;

-- Creamos un último trigger para comprobar que no nos pasamos de aforo en los
eventos
-- Como este trigger lanza un error, lo vamos a probar al final de todo el script
DELIMITER //
CREATE TRIGGER checkAforo
AFTER INSERT ON AsisteEvento
FOR EACH ROW
BEGIN
    DECLARE totalAttendees INT;
    DECLARE maxAforo INT;

    -- Get the total number of attendees for the event
    SELECT COUNT(*) INTO totalAttendees FROM AsisteEvento WHERE CodEvento =
NEW.CodEvento;

    -- Get the maximum capacity (Aforo) for the event's location
    SELECT Aforo INTO maxAforo FROM Ubicacion WHERE CodUbicacion = (SELECT
```

```
CodUbicacion FROM Evento WHERE CodEvento = NEW.CodEvento);

    -- Check if the total number of attendees exceeds the maximum capacity
    IF totalAttendees > maxAforo THEN
        -- reject the insertion or the line after to delete the row
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Exceeded maximum capacity for
the event.';
    END IF;
END;
//
DELIMITER ;



-- El trigger da error y no se ha insertado nada
select * from AsisteEvento;


/*--------------------------------------------------------------------------------
----------------------
Consultas, modificaciones, borrados y vistas con enunciado
--------------------------------------------------------------------------------
---------------------*/
-- Creamos una vista para obtener por cada evento que se celebra los ingresos
(suma de las entradas por cada asistente),
-- los gastos (cache de artistas y precio de alquiler de la ubicacion) y los
beneficios totales
CREATE VIEW GananciasPerdidasPorEvento AS
SELECT
    e.CodEvento,
    e.Descripcion,
    SUM(e.PrecioEntrada) AS Ganancias,
    u.Precio + COALESCE(SUM(p.Cache), 0) AS Gastos,
    SUM(e.PrecioEntrada) - (u.Precio + COALESCE(SUM(p.Cache), 0)) AS Beneficio
FROM
    Evento e
JOIN
    AsisteEvento ae ON e.CodEvento = ae.CodEvento
JOIN
    Ubicacion u ON e.CodUbicacion = u.CodUbicacion
LEFT JOIN
    Actividad a ON e.CodActividad = a.CodActividad
LEFT JOIN
    Participa p ON a.CodActividad = p.CodActividad
GROUP BY
    e.CodEvento, e.Descripcion, u.Precio;

-- Comprobamos la vista creada
SELECT * FROM GananciasPerdidasPorEvento;

-- Vista para ver el número de asistentes por cada tipo y subtipo de actividad
CREATE VIEW AsistentesPorTipoYSubtipoDeActividad AS
SELECT
    A.Tipo AS TipoActividad,
```

```
    S.Subtipo,
    COUNT(ASIST.Email) AS TotalAsistentes
FROM
    Actividad A
JOIN
    Evento E ON A.CodActividad = E.CodActividad
LEFT JOIN
    AsisteEvento ASIST ON E.CodEvento = ASIST.CodEvento
JOIN
    ActividadSubtipo ASUB ON A.CodActividad = ASUB.CodActividad
JOIN
    Subtipo S ON ASUB.CodSubtipo = S.CodSubtipo
GROUP BY
    A.Tipo, S.Subtipo;

SELECT * FROM AsistentesPorTipoYSubtipoDeActividad;


-- Vista para satisfacer el requisito del cliente de saber el caché total por
actividad sumando los cachés de los artistas que participan
CREATE VIEW CachesTotalesActividades AS
SELECT
    a.CodActividad,
    a.Nombre AS Actividad,
    a.Tipo,
    SUM(p.Cache) AS TotalCache
FROM
    Actividad a
JOIN
    Participa p ON a.CodActividad = p.CodActividad
GROUP BY
    a.CodActividad, a.Nombre, a.Tipo;

SELECT * FROM CachesTotalesActividades;

-- CONSULTAS

-- Consultas con las vistas

-- Obtener información sobre ganancias y pérdidas para cada evento
SELECT
    CodEvento,
    Descripcion,
    Ganancias,
    Gastos,
    Beneficio
FROM
    GananciasPerdidasPorEvento;

-- Ver el número total de asistentes por tipo y subtipo de actividad
SELECT
    TipoActividad,
    Subtipo,
    TotalAsistentes
```

```
FROM
    AsistentesPorTipoYSubtipoDeActividad;

-- Encontrar eventos con un tipo y subtipo específicos junto con sus ganancias y
pérdidas
SELECT
    G.CodEvento,
    G.Descripcion,
    G.Ganancias,
    G.Gastos,
    G.Beneficio,
    A.Tipo AS TipoActividad,
    S.Subtipo
FROM
    GananciasPerdidasPorEvento G
JOIN
    Evento E ON G.CodEvento = E.CodEvento
JOIN
    Actividad A ON E.CodActividad = A.CodActividad
JOIN
    ActividadSubtipo ASUB ON A.CodActividad = ASUB.CodActividad
JOIN
    Subtipo S ON ASUB.CodSubtipo = S.CodSubtipo
WHERE
    A.Tipo = 'concierto' AND S.Subtipo = 'Rock';

-- 1. Obtén todos los artistas y sus biografías
SELECT NombreArtista, Biografia
FROM Artista;

-- 2. Lista los eventos almacenados con sus descripciones y el nombre, dirección y
ciudad de sus correspondientes ubicaciones
SELECT e.CodEvento, e.Descripcion, e.Fecha, u.Nombre AS UbicacionNombre,
u.Direccion, u.Ciudad
FROM Evento e
JOIN Ubicacion u ON e.CodUbicacion = u.CodUbicacion;

-- 3. Para cada evento obtén el código de evento, el nombre y el número total de
asistentes
SELECT e.CodEvento, e.NombreEvento, COUNT(ae.Email) AS NumeroAsistentes
FROM Evento e
LEFT JOIN AsisteEvento ae ON e.CodEvento = ae.CodEvento
GROUP BY e.CodEvento, e.Descripcion;

-- 4. Lista las actividades de tipo "concierto" con los artistas que participan y
sus cachés
SELECT a.Nombre AS Actividad, a.Tipo, p.NombreArtista, p.Cache
FROM Actividad a
JOIN Participa p ON a.CodActividad = p.CodActividad
WHERE a.Tipo = 'concierto';

-- 5. Obtén el evento mayores ingresos y el nombre, dirección y ciudad de su
ubicación
SELECT e.CodEvento, e.NombreEvento, e.Descripcion, u.Nombre AS UbicacionNombre,
```

```
u.Direccion, u.Ciudad, SUM(e.PrecioEntrada) AS Ingresos
FROM Evento e
JOIN Ubicacion u ON e.CodUbicacion = u.CodUbicacion
JOIN AsisteEvento ae ON e.CodEvento = ae.CodEvento
GROUP BY e.CodEvento, e.Descripcion, u.Nombre, u.Direccion, u.Ciudad
ORDER BY Ingresos DESC
LIMIT 1;

-- 6. Obtén los beneficios de las distintas actividades por subtipo ordenadas de
mayor a menor
SELECT s.Subtipo, SUM(e.PrecioEntrada) AS IngresosTotales
FROM Subtipo s
JOIN ActividadSubtipo ast ON s.CodSubtipo = ast.CodSubtipo
JOIN Actividad a ON ast.CodActividad = a.CodActividad
JOIN Evento e ON a.CodActividad = e.CodActividad
JOIN AsisteEvento ae ON e.CodEvento = ae.CodEvento
GROUP BY s.Subtipo
ORDER BY IngresosTotales DESC;

-- 7. Lista los artistas que participaron en más de una actividad y lista sus
nombres
SELECT p.NombreArtista, COUNT(DISTINCT p.CodActividad) AS NumActividades
FROM Participa p
GROUP BY p.NombreArtista
HAVING NumActividades > 1;

-- 8. Lista los subtipos de "concierto" y el número de eventos en los que
participa cada subtipo
SELECT s.Subtipo, COUNT(DISTINCT ast.CodActividad) AS NumEventos
FROM Subtipo s
JOIN ActividadSubtipo ast ON s.CodSubtipo = ast.CodSubtipo
JOIN Actividad a ON ast.CodActividad = a.CodActividad
WHERE a.Tipo = 'concierto'
GROUP BY s.Subtipo;

-- 9. Lista todos los eventos con los nombres de los artistas que participan y sus
respectivos cachés ordenados por fecha de evento y caché
SELECT e.CodEvento, e.NombreEvento, e.Fecha, p.NombreArtista, p.Cache
FROM Evento e
JOIN Actividad a ON e.CodActividad = a.CodActividad
JOIN Participa p ON a.CodActividad = p.CodActividad
ORDER BY e.Fecha, p.Cache;

-- 10. Encuentra los subtipos de actividades asociados con los eventos de Granada
y su correspondiente aforo
SELECT s.Subtipo, u.Ciudad, COUNT(DISTINCT e.CodEvento) AS NumEvents, MAX(u.Aforo)
AS MaxCapacity
FROM Subtipo s
JOIN ActividadSubtipo ast ON s.CodSubtipo = ast.CodSubtipo
JOIN Actividad a ON ast.CodActividad = a.CodActividad
JOIN Evento e ON a.CodActividad = e.CodActividad
JOIN Ubicacion u ON e.CodUbicacion = u.CodUbicacion
WHERE u.Ciudad = 'Granada'
GROUP BY s.Subtipo, u.Ciudad;
```

```sql
-- Probamos trigger del aforo:
-- Probamos a pasarnos de aforo en algún evento
INSERT INTO AsisteEvento (CodEvento, Email) VALUES
    (2, 'juanPer34@example.com'),
    (1, 'M.gonzalez_99@example.com');
```