Objetivo

Diseño e implementación de una base de datos   
para gestionar una empresa de eventos en MySQL

Tarea final bases de datos relacionales

Autor: José Marcelo Morales Guamantaqui

## Enunciado:

Nuestra empresa, conocida como “ArteVida Cultural”, se dedica a la organización de eventos culturales únicos, que abarcan desde conciertos de música clásica hasta exposiciones de arte vanguardista, pasando por obras de teatro y conferencias. Es crucial gestionar la diversidad de eventos y detalles, así como las ganancias obtenidas. Para ello, es necesario llevar un registro adecuado de cada evento, de las actividades que se organizan, de los artistas que los protagonizan, las ubicaciones, la venta de entradas y el entusiasmo de los visitantes, quienes valorarán cada evento con un número del 0 al 5.

En nuestra empresa ofrecemos actividades con un nombre y un tipo: conciertos de música (clásica, pop, blues, soul, rock and roll, jazz, reggaeton, góspel, country, etc.), exposiciones, obras de teatro y conferencias. En cada actividad participa uno o varios artistas y tiene un coste (suma del caché de los artistas). El artista tiene un nombre, no tiene un caché fijo, y una breve biografía. Un artista puede participar en muchas actividades.

La ubicación tendrá un nombre (Teatro Maria Guerrero, Estadio Santiago Bernabeu, etc.), dirección, ciudad o pueblo, aforo, precio del alquiler y características. De cada evento necesitamos saber el nombre del evento (p.e. “VI festival de música clásica de Alcobendas”), la ubicación, el precio de la entrada, la fecha y la hora, así como una breve descripción. En un evento solo se realiza una actividad.

También tendremos en cuenta los asistentes a los eventos, de los que sabemos su nombre completo, teléfonos de contacto y email. Una persona puede asistir a más de un evento y a un evento pueden asistir varias personas, controlando que no se exceda el aforo de la ubicación.

Nos interesará realizar consultas de eventos por tipo de actividad, número de eventos de cada actividad, fecha con más eventos, ciudad con más eventos, actividades con un solo artista, ciudad con solo eventos de teatro, evento con más ceros en su valoración, etc.

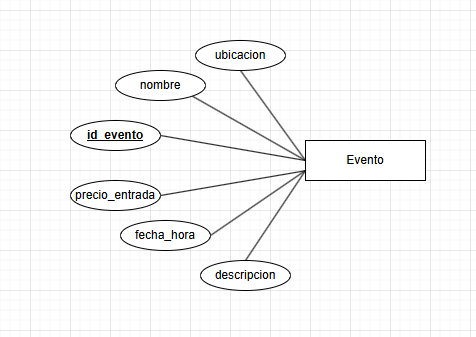
## **1. Diseño Conceptual**

#### 2.1 Identificación de Entidades:

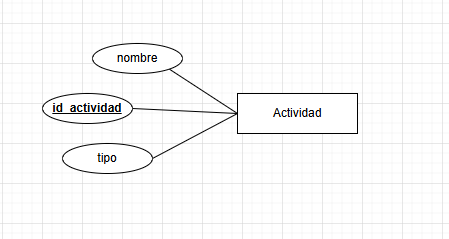
Basado en el enunciado del problema y las necesidades para gestionar la información de los eventos, se pueden identificar las siguientes entidades:

* Evento: es la entidad principal que queremos gestionar.
* Actividad: se realizan como parte de 1 evento.
* Artista: llevan a cabo las actividades.
* Asistente: personas que acuden al evento.
* Ubicación: localización donde se realiza el evento.

**Evento**: Esta tabla almacenaría información sobre cada evento específico. Se identifican los siguientes atributos:

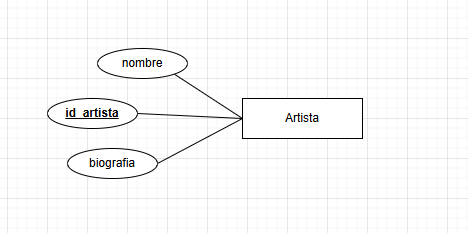


**Actividad**: Esta tabla detalla los tipos de actividades ofrecidas. Se identifican los siguientes atributos:

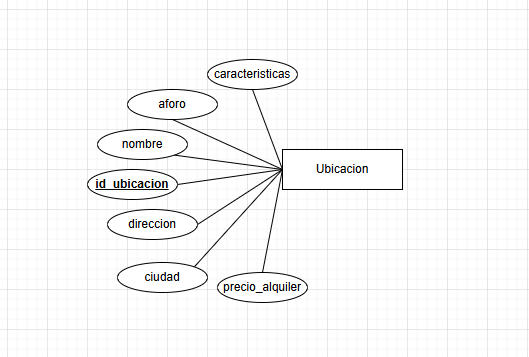


Nota: coste\_actividad dice ser la suma del cache de artistas, pero éste no es fijo. Por lo tanto se considera un atributo de la relación entre Actividad y Artista.

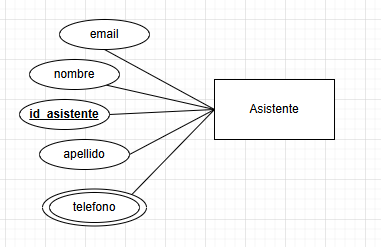
**Artista**: Esta tabla contendría información sobre los artistas que participan en los eventos. Se identifican los siguientes atributos:



**Ubicación**: Esta tabla almacenaría detalles sobre los lugares donde se realizan los eventos Se identifican los siguientes atributos:

****

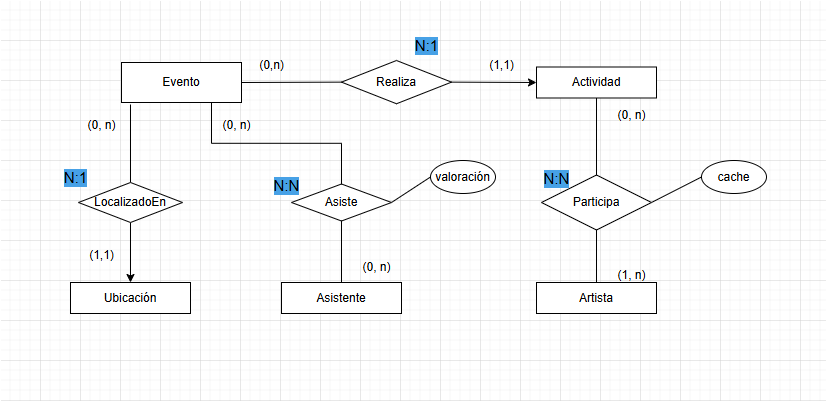
**Asistente**: Esta tabla contendría información sobre las personas que asisten a los eventos. Se identifican los siguientes atributos:



* Nota: teléfono es un atributo multivalorado porque el asistente puede tener varios.

### 2.1 Diagrama Entidad Relación

Basado en las entidades y relaciones identificadas se ha elaborado el siguiente diagrama Entidad-Relación (ER):

****

En el diagrama ER se establecen las siguientes relaciones:

**Realiza:** Representa larelación existente entre las 2 entidades **Evento** y **Actividad.**

* Cardinalidad: constituye una relación de **Varios-a-Uno**. Porque un mismo tipo de actividad se puede realizar en distintos eventos.
* Cardinalidad de máximos y mínimos: desde el punto de vista de Evento es (0, n) porque una actividad puede realizarse como mínimo en 0 eventos y como máximo en N eventos. Desde el punto de vista de Actividad es de (1, 1) porque un Evento puede constar como mínimo de 1 actividad y como máximo de 1 actividad por indicaciones del enunciado.

**Participa:** representa la relación existente entre las dos entidades **Actividad** y **Artista,** definida en el enunciado como “En cada actividad participa uno o varios artistas (…) Un artista pude participar en varias actividades”.

* Cardinalidad: constituye una relación de **Varios-a-Varios**.
* Atributos de relación: tiene un atributo de relación **cache,** porque en el enunciado se especifica que éste no es fijo. Por lo tanto depende de la participación de un artista en una actividad.
* Cardinalidad de máximos y mínimos: desde el punto de vista de Actividad es (0, n) porque un artista puede participar como mínimo en 0 actividades y como máximo en N actividades. Desde el punto de vista de Artista es (1, n) porque en una actividad puede participar como mínimo 1 artista y como máximo N artistas.

**Asiste:** representa la relación existente entre las entidades **Evento** y **Asistente,** está definida en el enunciado como “Una persona puede asistir a más de un evento y a un evento pueden asistir varias personas”.

* Cardinalidad: se considera una relación de **Varios-a-Varios.**
* Cardinalidad de máximos y mínimos: desde el punto de vista de Evento es (0, n) porque 1 asistente puede asistir como mínimo a 0 eventos y como máximo a N eventos. Desde el punto de vista de Asistente es (0, n) porque un evento puede tener como mínimo 0 asistentes y como máximo N asistentes.
* Atributos de relación: esta relación tendrá un atributo que es **Valoración,** que representa el número de estrellas dado por un asistente a un evento.

**LocalizadoEn:** representa la relación existente entre las dos entidades **Evento** y **Ubicación.**

* Cardinalidad**:** Se considera como una relación de **Varios-a-Uno** porque en una misma localización se pueden llevar a cabo varios eventos.
* Cardinalidad de máximos y mínimos: desde el punto de vista de Evento es (0, n) porque para una localización se puede tener 1 evento como mínimo y como máximo N eventos. Desde el punto de vista de Ubicación es (1, 1) porque un evento se puede localizar como mínimo en una ubicación y como máximo en una ubicación.

## 3. Diseño Lógico

En esta tercera fase del proyecto se aplican las técnicas aprendidas en el curso en el apartado “paso a tablas”, para transformar el diagrama entidad-relación obtenido en la fase 2 en su correspondiente modelo relacional, es decir, en un conjunto de relaciones con las claves primarias y ajenas.

Consideraciones:

* He introducido las claves ajenas (id\_actividad, idArtista, idUbicacion, idEvento, id\_asistente) para mantener la integridad referencial.
* La relación Varios-a-Varios que existe entre las tablas Actividad-Artista, y Asistente-Evento se resuelven creando tablas intermedias (Actividad\_Artista y Asistencia\_Evento). Estas tablas tendrán claves compuestas formadas por las claves ajenas que hacen referencia a las entidades originales.
* La tabla Actividad\_Artista incluye un atributo llamado cache\_artista para guardar lo que cobra cada artista para una actividad específica. Permitiendo calcular el costo total de la actividad como la suma de caches individuales.
* La tabla **Asistencia\_Evento** incluye el atributo **valoracion**, el cual describe la valoración dada por el asistente a cada evento al que ha asistido.
* Se definen claves ajenas para mantener la integridad referencial de las tablas. Las opciones **ON DELETE CASCADE** y **ON UPDATE CASCADE** se utilizan en algunas relaciones, lo que significa que si un registro se borra en la tabla principal o se cambia su clave primaria, los registros correspondientes en las tablas dependientes se actualizan en consecuencia.
* Añadimos una limitación de tipo **CHECK** a la table **Asistencia\_Evento** para asegurar que la valoración se mantiene en el rango de 0 a 5.
* Los tipos de datos se han elegido basado en la naturaleza de la información.

Teniendo en cuenta las anteriores consideraciones, las tablas del modelo lógico son:

|  |  |
| --- | --- |
| evento | |
| id\_evento | INT, PRIMARY KEY, AUTO\_INCREMENT |
| nombre | VARCHAR(255) |
| id\_ubicacion | INT NOT NULL |
| precio\_entrada | DECIMAL(10, 2) |
| fecha | DATE |
| hora | TIME |
| id\_actividad | INT NOT NULL |
| FOREIGN KEY (id\_ubicacion) REFERENCES Ubicacion(id\_ubicacion) ON DELETE CASCADE ON UPDATE CASCADE | |
| FOREIGN KEY (id\_actividad) REFERENCES Actividad(id\_actividad) ON DELETE CASCADE ON UPDATE CASCADE | |

|  |  |
| --- | --- |
| ubicación | |
| id\_ubicacion | INT, PRIMARY KEY, AUTO\_INCREMENT |
| nombre | VARCHAR(255) |
| direccion | VARCHAR(255) |
| ciudad\_pueblo | VARCHAR(255) |
| aforo | INT |
| precio\_alquiler | DECIMAL(10, 2) |
| caracteristicas | TEXT |

|  |  |
| --- | --- |
| artista | |
| id\_artista | INT, PRIMARY KEY, AUTO\_INCREMENT |
| nombre | VARCHAR(255) |
| biografia | (TEXT) |

|  |  |
| --- | --- |
| actividad | |
| id\_actividad | INT, PRIMARY KEY, AUTO\_INCREMENT |
| nombre | VARCHAR(255) |
| tipo | VARCHAR(50) |
| coste | DECIMAL(10, 2) |

|  |  |
| --- | --- |
| asistente | |
| id\_asistente | INT, PRIMARY KEY, AUTO\_INCREMENT |
| nombre\_completo | VARCHAR(255) |
| email | VARCHAR(255) |

|  |  |
| --- | --- |
| telefono\_asistente | |
| id\_asistente | INT NOT NULL |
| numero\_telefono | VARCHAR(20) NOT NULL |
| FOREIGN KEY (id\_asistente) REFERENCES Asistente(id\_asistente) ON DELETE CASCADE ON UPDATE CASCADE | |
| PRIMARY KEY (id\_asistente, numero\_telefono) | |

|  |  |
| --- | --- |
| actividad\_artista | |
| id\_actividad | INT NOT NULL |
| id\_artista | INT NOT NULL |
| cache\_artista | DECIMAL (10, 2) |
| FOREIGN KEY (id\_actividad) REFERENCES Actividad(id\_actividad) ON DELETE CASCADE ON UPDATE CASCADE | |
| FOREIGN KEY (id\_artista) REFERENCES Artista(id\_artista) ON DELETE CASCADE ON UPDATE CASCADE | |
| PRIMARY KEY (id\_actividad, id\_artista) | |

|  |  |
| --- | --- |
| asistencia\_evento | |
| id\_asistente | INT NOT NULL |
| id\_evento | INT NOT NULL |
| valoracion | INT CHECK (valoracion BETWEEN 0 and 5) |
| FOREIGN KEY (id\_asistente) REFERENCES Asistente(id\_asistente) ON DELETE CASCADE ON UPDATE CASCADE | |
| FOREIGN KEY (id\_evento) REFERENCES Evento(id\_evento) ON DELETE CASCADE ON UPDATE CASCADE | |
| PRIMARY KEY (id\_asistente, id\_evento) | |

## 4. Implementación

Script copiado:

/\*

SCRIPT TAREA FINAL.

Asignatura: BBDD SQL

Nombre del Autor: Morales Guamantaqui, José Marcelo

Nombre de la base de datos: empresa\_eventos

Fecha: 2025-03-17

\*/

/\* ------------------------------------------------------------------------------------------------

Definición de la estructura de la base de datos

--------------------------------------------------------------------------------------------------\*/

USE Empresa\_Eventos;

-- Crear Tabla: ubicacion

CREATE TABLE IF NOT EXISTS ubicacion (

id\_ubicacion INT PRIMARY KEY AUTO\_INCREMENT,

nombre VARCHAR(255),

direccion VARCHAR(255),

ciudad\_pueblo VARCHAR(255),

aforo INT,

precio\_alquiler DECIMAL(10, 2),

caracteristicas TEXT

);

-- Crear Tabla: actividad

CREATE TABLE IF NOT EXISTS actividad (

id\_actividad INT PRIMARY KEY AUTO\_INCREMENT,

nombre VARCHAR(255),

tipo VARCHAR(50),

coste DECIMAL(10, 2)

);

-- Crear Tabla: artista

CREATE TABLE IF NOT EXISTS artista (

id\_artista INT PRIMARY KEY AUTO\_INCREMENT,

nombre VARCHAR(255),

biografia TEXT

);

-- Crear Tabla: asistente

CREATE TABLE IF NOT EXISTS asistente (

id\_asistente INT PRIMARY KEY AUTO\_INCREMENT,

nombre\_completo VARCHAR(255),

email VARCHAR(255)

);

-- Crear Tabla: evento

CREATE TABLE IF NOT EXISTS evento (

id\_evento INT PRIMARY KEY AUTO\_INCREMENT,

nombre VARCHAR(255),

id\_ubicacion INT NOT NULL,

precio\_entrada DECIMAL(10, 2),

fecha DATE,

hora TIME,

id\_actividad INT NOT NULL,

FOREIGN KEY (id\_ubicacion) REFERENCES ubicacion(id\_ubicacion) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (id\_actividad) REFERENCES actividad(id\_actividad) ON DELETE CASCADE ON UPDATE CASCADE

);

-- Crear Tabla: telefono\_asistente

CREATE TABLE IF NOT EXISTS telefono\_asistente (

id\_asistente INT NOT NULL,

numero\_telefono VARCHAR(20) NOT NULL,

FOREIGN KEY (id\_asistente) REFERENCES asistente(id\_asistente) ON DELETE CASCADE ON UPDATE CASCADE,

PRIMARY KEY (id\_asistente, numero\_telefono)

);

-- Crear Tabla: actividad\_artista

CREATE TABLE IF NOT EXISTS actividad\_artista (

id\_actividad INT NOT NULL,

id\_artista INT NOT NULL,

cache\_artista DECIMAL(10, 2),

FOREIGN KEY (id\_actividad) REFERENCES actividad(id\_actividad) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (id\_artista) REFERENCES artista(id\_artista) ON DELETE CASCADE ON UPDATE CASCADE,

PRIMARY KEY (id\_actividad, id\_artista)

);

-- Crear Tabla: asistencia\_evento

CREATE TABLE IF NOT EXISTS asistencia\_evento (

id\_asistente INT NOT NULL,

id\_evento INT NOT NULL,

valoracion INT CHECK (valoracion BETWEEN 0 AND 5),

FOREIGN KEY (id\_asistente) REFERENCES asistente(id\_asistente) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (id\_evento) REFERENCES evento(id\_evento) ON DELETE CASCADE ON UPDATE CASCADE,

PRIMARY KEY (id\_asistente, id\_evento)

);

/\*-----------------------------------------------------------------------------------------------------------

Trigger: revisa que cada vez que se añade un evento nuevo. Debe tener correspondiente ubicación y actividad.

-------------------------------------------------------------------------------------------------------------\*/

DELIMITER //

CREATE TRIGGER check\_ubicacion\_actividad

BEFORE INSERT ON evento

FOR EACH ROW

BEGIN

-- Declaramos variables para comprobar existencia de claves ajenas.

DECLARE ubicacion\_exists INT;

DECLARE actividad\_exists INT;

-- Check: si id\_ubicacion existe en la tabla ubicacion

SELECT COUNT(\*) INTO ubicacion\_exists

FROM ubicacion

WHERE id\_ubicacion = NEW.id\_ubicacion;

-- Check: si id\_actividad existe en la tabla actividad

SELECT COUNT(\*) INTO actividad\_exists

FROM actividad

WHERE id\_actividad = NEW.id\_actividad;

-- Si el id\_ubicacion no existe, devolver un mensaje

IF ubicacion\_exists = 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Error: Trigger message: el valor de ubicacion\_id indicado no existe en la tabla ubicacion.';

END IF;

-- Si id\_actividad no existe, devolver un mensaje

IF actividad\_exists = 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Error: Trigger message: el valor de actividad\_id indicado no existe en la tabla actividad.';

END IF;

END;

// DELIMITER ;

/\*------------------------------------------------------------------------------------------------------

Inserción de Datos: se ha hecho siguiendo procedimiento descrito en el documento "Cargar Datos Workbench"

Los archivos CSV están en la carpeta RAW\_DATA del directorio Raíz.

-------------------------------------------------------------------------------------------------------\*/

SELECT @@GLOBAL.secure\_file\_priv;

-- Cargamos tabla ACTIVIDAD

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/actividad\_raw.csv'

INTO TABLE actividad

CHARACTER SET utf8

FIELDS TERMINATED BY ';'

LINES TERMINATED BY '\n'

IGNORE 1 rows

(nombre, tipo, coste);

-- Carga de tabla Ubicacion

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/ubicacion\_raw.csv'

INTO TABLE ubicacion

CHARACTER SET utf8

FIELDS TERMINATED BY ';'

LINES TERMINATED BY '\n'

IGNORE 1 rows

(nombre, direccion, ciudad\_pueblo, aforo, precio\_alquiler, caracteristicas);

-- Carga de tabla Asistente

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/asistente\_raw.csv'

INTO TABLE asistente

CHARACTER SET utf8

FIELDS TERMINATED BY ';'

LINES TERMINATED BY '\n'

IGNORE 1 rows

(nombre\_completo, email);

-- Carga de tabla Artista

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/artista\_raw.csv'

INTO TABLE artista

CHARACTER SET utf8

FIELDS TERMINATED BY ';'

LINES TERMINATED BY '\n'

IGNORE 1 rows

(nombre, biografia);

-- Carga de tabla Evento

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/evento\_raw.csv'

INTO TABLE evento

CHARACTER SET utf8

FIELDS TERMINATED BY ';'

LINES TERMINATED BY '\n'

IGNORE 1 rows

( nombre, id\_ubicacion, precio\_entrada, fecha,hora,id\_actividad);

-- Carga de tabla actividad\_artista

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/actividad\_artista\_raw.csv'

INTO TABLE actividad\_artista

CHARACTER SET utf8

FIELDS TERMINATED BY ';'

LINES TERMINATED BY '\n'

IGNORE 1 rows

(id\_actividad, id\_artista, cache\_artista);

-- Carga de tabla asistencia\_evento

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/asistencia\_evento\_raw.csv'

INTO TABLE asistencia\_evento

CHARACTER SET utf8

FIELDS TERMINATED BY ';'

LINES TERMINATED BY '\n'

IGNORE 1 rows

(id\_asistente, id\_evento, valoracion);

-- Carga de tabla telefono\_asistente

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/telefono\_asistente\_raw.csv'

INTO TABLE telefono\_asistente

CHARACTER SET utf8

FIELDS TERMINATED BY ';'

LINES TERMINATED BY '\n'

IGNORE 1 rows

(id\_asistente, numero\_telefono);

-- Intentamos insertar un evento para probar el trigger

Insert into evento (nombre, id\_ubicacion, precio\_entrada, fecha, hora, id\_actividad)

VALUES ('test\_event', 33393, 33, '2025-12-01', '10:00', 6666);

-- El trigger funciona devolviendo un mensaje claro

/\*------------------------------------------------------------------------------------------------------

Consultas, modificaciones, borrados y vistas con enunciado

-------------------------------------------------------------------------------------------------------\*/

-- VISTA: creamos la vista que enlaza las tablas: actividad - actividad\_artista - artista para sacar el coste de actividad

CREATE VIEW vw\_coste\_actividad AS

SELECT

A.id\_actividad,

A.nombre AS nombre\_actividad,

A.tipo AS tipo\_actividad,

SUM(ACT.cache\_artista) AS coste\_actividad,

COUNT(ART.id\_artista) AS numero\_artistas

FROM actividad AS A

JOIN actividad\_artista AS ACT

ON A.id\_actividad = ACT.id\_actividad

JOIN artista AS ART

ON ACT.id\_artista = ART.id\_artista

GROUP BY A.id\_actividad, A.nombre, A.tipo;

-- Esto nos permite actualizar el coste en la tabla actividad.

UPDATE actividad A

JOIN vw\_coste\_actividad W

ON A.id\_actividad = W.id\_actividad

SET A.coste = W.coste\_actividad;

-- Número de eventos por tipo\_actividad

SELECT

A.tipo AS tipo\_actividad,

COUNT(E.id\_evento) AS num\_eventos

FROM evento AS E

JOIN actividad AS A

ON A.id\_actividad = E.id\_actividad

GROUP BY A.tipo;

-- Eventos clasificados de más caro a más barato (precio entrada)

SELECT

nombre,

precio\_entrada,

DENSE\_RANK() OVER (ORDER BY precio\_entrada) AS rank\_entrada

FROM evento AS E

ORDER BY DENSE\_RANK() OVER (ORDER BY precio\_entrada);

-- Número de eventos por año y coste total

SELECT

YEAR(fecha),

COUNT(id\_evento) AS numero\_eventos,

SUM(A.coste) AS coste\_eventos

FROM evento AS E

JOIN actividad AS A

ON E.id\_actividad = A.id\_actividad

GROUP BY YEAR(fecha)

ORDER BY YEAR(fecha);

-- Artistas clasificados por suma de caché para ver quién ha cobrado más

SELECT

ART.nombre AS nombre\_artista,

SUM(ACT.cache\_artista) AS total\_cache,

DENSE\_RANK() OVER (ORDER BY SUM(ACT.cache\_artista) DESC) AS cache\_ranking

FROM actividad\_artista AS ACT

JOIN artista AS ART

ON ART.id\_artista = ACT.id\_artista

GROUP BY ART.nombre;

-- Eventos clasificados por media de valoración: si valoracion\_media IS NULL significa que no ha habido asistentes

SELECT

DENSE\_RANK() OVER (ORDER BY AVG(AE.valoracion) DESC) AS posicion,

E.nombre AS nombre\_evento,

AVG(AE.valoracion) AS valoracion\_media

FROM evento AS E

LEFT JOIN asistencia\_evento AS AE

ON E.id\_evento = AE.id\_evento

GROUP BY E.nombre;

-- Listado de actividades realizadas con primera y última fecha

SELECT

ACT.id\_actividad,

ACT.nombre AS nombre\_actividad,

MIN(EV.fecha) AS primera\_vez\_actividad,

MAX(EV.fecha) AS ultima\_vez\_actividad

FROM actividad AS ACT

LEFT JOIN evento AS EV

ON ACT.id\_actividad = EV.id\_actividad

WHERE EV.fecha IS NOT NULL

GROUP BY ACT.id\_actividad, ACT.nombre

ORDER BY ACT.id\_actividad;

-- Teléfonos y email de los asistentes a determinados eventos

SELECT

AS\_EV.id\_evento,

ATT.nombre\_completo,

TEL\_AS.numero\_telefono,

ATT.email

FROM asistencia\_evento AS AS\_EV

JOIN telefono\_asistente AS TEL\_AS

ON AS\_EV.id\_asistente = TEL\_AS.id\_asistente

JOIN asistente AS ATT

ON ATT.id\_asistente = TEL\_AS.id\_asistente

WHERE id\_evento IN (4, 7);

-- Actividades donde ha participado más de 1 artista

SELECT

id\_actividad,

nombre\_actividad,

coste\_actividad,

numero\_artistas

FROM vw\_coste\_actividad

WHERE numero\_artistas > 1;

-- Borrado de actividades no realizadas nunca

SET SQL\_SAFE\_UPDATES = 0;

DELETE A

FROM actividad AS A

LEFT JOIN evento AS E

ON A.id\_actividad = E.id\_actividad

WHERE E.id\_evento IS NULL;

SET SQL\_SAFE\_UPDATES = 1;