Tarea Bases de Datos SQL

Autor: *Álvaro Vilela Nova*

1. Especificación de requisitos

Se nos pide analizar la situación del enunciado para poder diseñar una base de datos adecuada. Está claro que la base de datos necesitará guardar los siguientes datos básicos:

- Eventos organizados por la empresa.
- Ubicaciones donde se organizan eventos.
- Gente que protagoniza el evento, por ejemplo, artistas o conferenciantes. Nos referiremos a ellos de forma genérica como artistas.
- Personas que asisten al evento.

El enunciado nos da indicaciones de ciertos atributos que debemos poder guardar en nuestra base de datos como mínimo. Sin embargo, hay ciertas consultas o datos que parece intuitivo o razonable que el usuario quiera poder acceder con facilidad. Estos datos están en el modelo conceptual más adelante en el documento. Algunos de estos datos son:

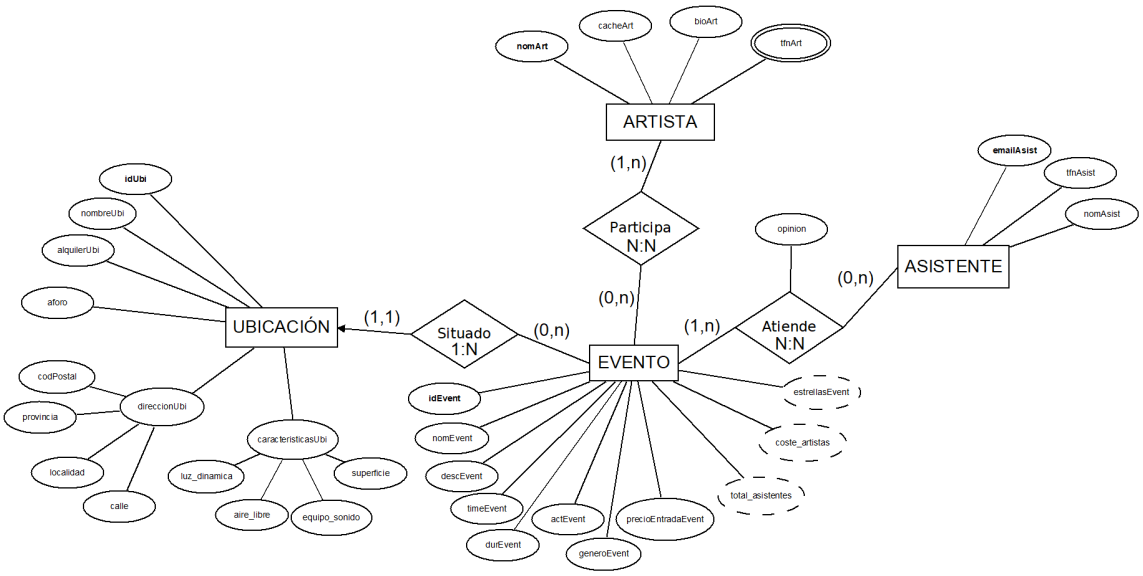
- Medios disponibles en las ubicaciones.
- Datos acerca de los ingresos, gastos y beneficios de los eventos.
- Valoración general de los asistentes de un evento.

2. Diseño conceptual

Ya que tenemos una idea coherente de qué datos debemos almacenar en nuestra base de datos. Ahora, formalizaremos el Modelo Entidad-Relación.

Modelo E-R

Aquí tenemos el diagrama resultante del proceso de análisis de requisitos. A partir de este diagrama diseñaremos la base de datos.



Las claves primarias están marcadas en negrita porque el software utilizado (Día) no tiene soporte para esta fuente con subrayado. Lo mismo sucede con Markdown; a lo largo de este documento, las claves primarias están marcadas añadiendo de forma explícita **(clave primaria)** o escritas en *cursiva*.

Con esta estructura garantizamos que nuestra base de datos pueda contener todos los datos que debemos administrar usando las entidades **UBICACION**, **EVENTO**, **ARTISTA** Y **ASISTENTE**. Podemos comprobar que todas la relaciones utilizadas son binarias, lo que simplifica el modelo. Tampoco tenemos ciclos de relaciones y así evitamos posibles redundancias. Tenemos varios tipos de atributos: simples, compuestos, multivalorados y derivados. A continuación, se da una descripción de estas entidades, relaciones y atributos.

Al final de esta sección se explican algunas consideraciones y asunciones que hemos tenido en cuenta, además de conceptos que no hemos utilizado en nuestro modelo.

Entidades y atributos

Vamos a tener cuatro entidades en total:

- **Evento:** evento que organiza la empresa. Contiene datos relativos a las finanzas relacionadas con cada evento individual y sus características.
  - ***idEvent* (clave primaria):** clave primaria de la entidad. Usamos esto en lugar del nombre porque podríamos tener eventos recurrentes, por ejemplo, conciertos organizados todos los años.
  - **nomEvent (simple):** nombre del evento.
  - **descEvent (simple):** breve descripción del evento.
  - **timeEvent (simple):** fecha y hora a la que comienza el evento.
  - **durEvent (simple):** número de días que dura el evento. Algunos eventos musicales o convenciones científicas pueden durar varios días.
  - **actEvent (simple):** el tipo de actividad en la que consiste el evento.
  - **generoEvent (simple):** el género del evento, que puede ser distintos valores dependiendo del tipo de actividad; por ejemplo, en un concierto este atributo sería el género de la música, mientras que en una conferencia sería el tema (divulgación científica, finanzas, etc).
  - **precioEntradaEvent (simple):** el coste de la entrada al evento.
  - **total_asistentes (derivado):** número de asistentes al evento, equivaldría al número de entradas vendidas para el evento.
  - **coste_artistas (derivado):** la suma del cache de los artistas que participan en el evento.
  - **estrellasEvent (derivado):** la media de las opiniones dadas por los asistentes al evento. (El atributo *opinion* está explicado más adelante en el apartado de relaciones).
- **Artista:** artista o conferenciante que realiza las actividades del evento.
  - ***nomArt* (clave primaria):** el nombre artístico o del conferenciante, que creo razonable que será único.
  - **cacheArt (simple):** el caché del artista.
  - **bioArt (simple):** una corta biografía o descripción del artista.
  - **tfnArt (multivalorado):** el telefono de contacto del artista. Es razonable pensar que podría ser multivalorado, por ejemplo, podemos tener el teléfono personal del artista y el de su manager o discográfica.

- **Asistente:** persona que compra una entrada para asistir a eventos que organizamos.
  - ***emailAsist* (clave primaria):** el email del asistente. Consideramos que las personas deben registrarse con un email en una página para comprar la entrada y recibirla por correo electrónico. Sería razonable que el correo entonces identificará a los asistentes de forma única.
  - **tfnAsist (simple):** el teléfono de contacto del asistente. No lo consideramos multivalorado como el del artista, pues parece bastante estándar en la industria no pedir varios teléfonos a la hora de registrarse.
  - **nomAsist (simple):** el nombre y apellidos del asistente. No usamos esto como clave primaria por la posibilidad de dos personas con el mismo nombre y apellidos.
- **Ubicación:** ubicación disponible para organizar eventos.
  - ***ibUbi* (clave primaria):** un id con el que identificamos la ubicación, usamos esto en lugar del nombre para evitar entradas duplicadas por posibles errores ortográficos.
  - **nombreUbi (simple):** nombre de la ubicación.
  - **alquilerUbi (simple):** coste de alquiler de la ubicación por día.
  - **direccionUbi (compuesto):** dirección de la ubicación.
    - **codPostal (simple):** código postal de la dirección.
    - **provincia (simple):** provincia de la dirección.
    - **localidad (simple):** ciudad o pueblo donde está la ubicación. Consideramos que podría haber varias localidades con un mismo nombre, pero gracias a que tenemos tanto provincia como codPostal podemos discriminar entre datos.
    - **calle (simple):** la calle y número de la ubicación. Podría ser compuesto, pero consideramos que el atributo simple es más cómodo en su uso.
  - **aforo (simple):** número máximo de personas que puede acoger la ubicación.
  - **característicasUbi (compuesto):** algunas características o equipamiento que puede tener la ubicación.
    - **luz_dinámica (simple):** si la ubicación está equipada con iluminación como focos o luces de colores.
    - **aire_libre (simple):** si la ubicación tiene areas al aire libre.
    - **equipo_sonido (simple):** si la ubicación incluye equipamiento para configurar y balancear el sonido de conciertos.
    - **superficie (simple):** metros cuadrados utilizables en la ubicación.

Relaciones y cardinalidades

Las entidades se van a relacionar de la siguiente manera:

- **Situado (Evento -> Ubicación):** un evento está situado en una ubicación. 1:N
  - **Cardinalidad (1,1).** Cada evento, como mínimo, debe estar situado en un sitio y no puede estar situado en varios sitios distintos a la vez.
  - **Cardinalidad (0,n).** En cada ubicación de nuestra base de datos podemos, como mínimo, no tener un evento todavía organizado o, como máximo, podemos tener organizados varios eventos.
- **Participa (Evento - Artista):** artistas participan en eventos. N:N
  - **Cardinalidad (1,n).** Como mínimo, en un evento debe participar un artista; como máximo, pueden participar varios artistas.

    - **Cardinalidad (0,n).** Como mínimo, un artista que tenemos registrado aún no está en un evento organizado; como máximo, el artista está registrado en varios eventos.
- **Atiende (Evento - Asistente):** los asistentes asisten a eventos. N:N
    - **Cardinalidad (0,n).** Como mínimo, un evento puede no tener ni un solo asistente; como máximo, un evento puede ser asistido por muchas personas.
    - **Cardinalidad (1,n).** Como mínimo, un asistente asistirá a un evento (si no, no lo tendríamos registrado en nuestra base de datos); como máximo, un asistente puede asistir a muchos eventos.
    - **opinion (simple)** es un atributo de la relación que sería la valoración en formato de 5 estrellas que da el asistente al evento. Notar que un asistente podría no dar valoración al evento.

Dominios

Aqui tenemos las tablas que nos dan los dominios sobre los que están definidos los atributos de cada entidad, con sus tipos de dato:

Evento	Dominio	Tipo de dato
idEvent	Conjunto de números naturales	Valor numérico int autoincremental
nomEvent	Conjunto de posibles nombres de eventos	Cadena caracteres
descEvent	Descripciones de eventos	Cadena caracteres
timeEvent	Conjunto de fechas	TimeStamp
durEvent	Conjunto de números naturales no-negativos	Valor numérico int
actEvent	Conjunto de actividades que organizamos	Cadena caracteres
generoEvent	Conjunto de géneros de música o temas	Cadena caracteres
precioEntradasEvent	Conjunto de números reales positivos	Valor numérico
total_asistentes	Conjunto de números naturales	Valor numérico int
coste_artistas	Conjunto de números reales positivos	Valor numérico
estrellasEvent	Conjunto de números reales entre 0 y 5	Valor numérico

Artista	Dominio	Tipo de dato
nomArt	Conjunto de nombres de artistas	Cadena de caracteres

cacheArt	Conjunto de caché de los artistas	Valor numérico
bioArt	Conjunto de biografías de artistas	Cadena caracteres
tfnArt	Conjunto de conjuntos de telefonos de artistas	Cadenas de caracteres (puede haber símbolos relevantes en el número de telefono, como + o () en prefijos internacionales) en una tabla relacionada con esta

Asistente	Dominio	Tipo de dato
emailAsist	Conjunto de emails	Cadena caracteres
tfnAsist	Conjunto de números de teléfono	Cadena caracteres (por el mismo motivo que en el caso de los artistas)
nomAsist	Conjunto de nombres completos	Cadena caracteres

Ubicación	Dominio	Tipo de dato
idUbi	Conjunto de claves numéricas	Valor numérico int autoincremental
nombreUbi	Conjunto de nombres de ubicaciones	Cadena caracteres
alquilerUbi	Conjunto de valores reales no-negativos	Valor numérico
aforo	Conjunto de números naturales	Valor numérico
direccionUbi		(Atributo compuesto)
codPostal	Conjunto de códigos postales	Cadena caracteres (en caso de tener un código postal que comience por 0)
provincia	Conjunto de provincias de España	Cadena caracteres
localidad	Conjunto de localidades de España	Cadena caracteres
calle	Conjunto de calles de España	Cadena caracteres
caracteristicasUbi		(Atributo compuesto)
luz_dinámica	Conjunto {0,1}	Booleano
aire_libre	Conjunto {0,1}	Booleano

equipo_sonido	Conjunto {0,1}	Booleano
superficie	Conjunto de números reales positivos	Valor numérico

Ahora, los dominios de las relaciones. De la misma forma que se muestra en la diapositiva 11 del archivo *03-Modelo Relacional*.

Situado	**Dominio**
idEvent	Conjunto de identificadores de los eventos
idUbi	Conjunto de identificadores de las ubicaciones

Participa	**Dominio**
idEvent	Conjunto de identificadores de los eventos
nomArt	Conjunto de nombres de los artistas

Atiende	**Dominio**
idEvent	Conjunto de identificadores de los eventos
emailAsist	Conjunto de emails de los asistentes
opinion	Conjunto de números naturales entre 0 y 5

Conceptos no expresados en la modelización

Algunos conceptos que hemos estudiado en clase no están presentes en nuestro modelo. Estos son:

- Relaciones ternarias.
- Relaciones reflexivas.
- Entidades débiles.
- Relaciones débiles.
- Atributos discriminadores en entidades débiles.
- Especializaciones.
- Generalizaciones.
- Relaciones uno a uno.

Estos conceptos están ausentes en nuestro modelo por diversos motivos, principalmente que tal como lo tenemos diseñado tiene una simplicidad suficiente. No tenemos relaciones uno a uno pues parece que ninguna de las entidades de relaciona uno a uno con otra. Se podría haber usado especializaciones para definir una entidad **PERSONA** y separarla en **ARTISTAS** y **ASISTENTES**; sin embargo, parece que poseen pocas cualidades en común y acceder a los datos a través de una especialización complicaría el acceso, de otra forma sencillo, a nuestros datos.

Algunos detalles añadidos y consideraciones

Primero, mencionar que en versiones anteriores del modelo relacional incluía algunos atributos derivados que generaban redundancia, por ejemplo: un atributo **ingresos**, que era calculado usando **precioEntradasEvent** y **total_asistentes**; un atributo **coste_total** que sumaba el precio de alquilar la ubicación todos los días y el coste de los

artistas; finalmente, otro atributo **balanceEvent**, que era calculado usando **coste_total** e **ingresos**. Mencionamos en clase en las recomendaciones para el diseño de *02 - Modelo Entidad Relación* lo siguiente:

> *Atributos derivados, si un atributo se puede extraer de otros dos, no hace falta poner el derivado*

Consideré si mantener esos atributos para tener esos datos fácilmente accesible merecía la pena el riesgo de redundancia y la posibilidad de reducir la eficiencia de la base de datos. Finalmente, decidí eliminarlos y simplemente calcularlos como consultas. Se pueden ver los modelo anterior haciendo click aquí y aquí. Mantenerlos actualizados con cada operación de escritura requería muchos triggers (y ya hay bastantes implementados en el código). Algunos atributos derivados que he añadido son calculados a partir de atributos que se encuentran en otras entidades, por tanto, no los considero redundantes.

He añadido algunos atributos a la entidad ubicación, en particular, los atributos que componen **caracteristicasUbi**. Podría haber hecho **carácterísticasUbi** una cadena de caracteres en la que escribiriamos "tags" que indiquen ciertas características y después usariamos consultas para buscar ciertas características. He creído conveniente utilizar esta forma para mostrar como funcionan los atributos booleanos en el sistema con el que trabajamos; si bien es cierto que quizás la otra alternativa podría ser más óptima.

He considerado también los telefonos de los artistas como atributos multivalorados para mostrar como funcionan, si bien podría haber asumido que tenemos un único número de contacto. También he considerado que el alquiler de una ubicación es siempre el mismo por día, el alquiler podría cambiar con el tiempo y si lo cambiásemos en la base de datos los costes totales de eventos que se organizaron antes del cambio del alquiler se verían afectados; quiero destacar que es una asunción que no cambia, de no ser así, podríamos simplemente poner **alquilerUbi** como un atributo de la relación **Situado**. Como ya se ilustra el uso de este tipo de atributos en la relación **Atiende**, he decidido no complicar más el modelo. Ya se mencionó más arriba que los nombres de los artistas asumimos que serán únicos.

3. Diseño lógico

En este apartado vamos a denotar las claves primarias usando *cursiva*, pues Markdown no tiene soporte para subrayado. Las claves ajenas se van a señalar usando ***cursiva y negrita***. En el dibujo donde aparece el paso a tablas con todas las flechas relacionando las distintas claves se utiliza la notación que aprendimos en clase, así que es mejor fiarse del dibujo.

Paso a tablas

Este es el paso a tablas de las entidades.

EVENTO(*idEvent*, nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent, precioEntradaEvent, total_asistentes, coste_artistas, coste_total, estrellasEvent, ***cod_Ubi***)

UBICACION(*idUbi*, nombreUbi, alquilerUbi, aforo, direccionUbi, caracteristicasUbi)

ARTISTA(*nomArt*, cacheArt, bioArt)

tfnArt(***NombreArtista***, tfnArtista, tfnManager)

ASISTENTE(*emailAsist*, tfnAsist, nomAsist)

Notar que la clave primaria en la tabla **tfnArt** es ***NombreArtista*** que a su vez es la clave ajena que relaciona los números de telefono con los artistas, es decir, hay una relación uno a uno entre la tabla **ARTISTAS** y la tabla **tfnArt**. Notar también que como **direccionUbi** y **caracteristicasUbi** son variables compuestas, en realidad tenemos:

UBICACION(*idUbi*, nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad, calle, luz_dinamica, aire_libre, equipo_sonido, superficie)

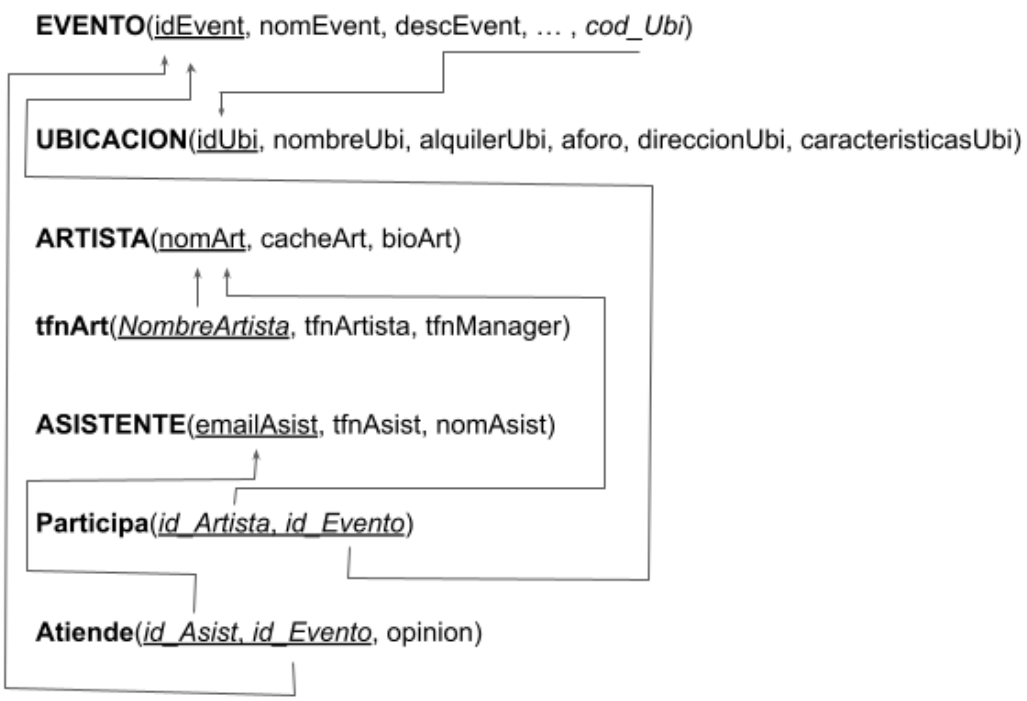
Este es el paso a tablas de las relaciones.

Situado: por ser 1:N se añade un atributo que será clave ajena ***cod_Ubi*** a la entidad **EVENTO**. Ya está incluído más arriba.

Participa(***id_Artista***, ***id_Evento***)

Atiende(***id_Asist***, ***id_Evento***, opinion)

Ahora, tenemos que escribir esto y dibujar las flechas que relacionan las tablas, sus claves primarias y ajenas entre sí.



El diagrama esquema generado por MySQL es bastante raro y he tenido cantidad de problemas para conseguir siquiera generarlo (supongo que por alguna incompatibilidad con mi SO). Se puede ver aquí.

4. Implementación

El código incluye 8 Triggers que buscan garantizar relaciones correctas entre los datos, además de mantener actualizados algunos atributos derivados. En la parte final del código *Consultas, modificaciones, borrados y vistas con enunciado* se comprueba el buen funcionamiento de los triggers; además, incluye algunas consultas que pueden resultar interesantes para quien vaya a gestionar el negocio usando esta base de datos. Estas incluyen:

- Beneficios obtenidos por cada evento.
- Listados que nos indican qué pagos debemos hacer a artistas y ubicaciones.
- Algunos listados que nos permiten ver qué tipo de eventos tienen más o menos éxito.
- Cómo de activos son y cuánto gastan los asistentes que tenemos registrados en la base de datos.

Entre las consultas se incluyen algunas más sencillas, otras más complejas y una en la que se utiliza una vista.

```
/* ----------------------------------------------------------------------------
--------
Álvaro Vilela Nova
ArteVida Cultural
----------------------------------------------------------------------------
-------------*/
/* ----------------------------------------------------------------------------
-------------
Definición de la estructura de la base de datos
----------------------------------------------------------------------------
------------*/
DROP DATABASE IF EXISTS ARTEVIDA;

CREATE DATABASE ARTEVIDA;
USE ARTEVIDA;

DROP TABLE IF EXISTS UBICACION;
DROP TABLE IF EXISTS ASISTENTE;
DROP TABLE IF EXISTS ARTISTA;
DROP TABLE IF EXISTS EVENTO;
DROP TABLE IF EXISTS tfnArt;
DROP TABLE IF EXISTS Atiende;
DROP TABLE IF EXISTS Participa;


CREATE TABLE UBICACION (
    idUbi SMALLINT auto_increment,
    nombreUbi VARCHAR(100),
    alquilerUbi FLOAT NOT NULL, -- Este es un dato especialmente importante, lo usamos
para atributos derivados y es muy relevante a la hora de organizar eventos. No puede
ser nulo.
    aforo SMALLINT,
    codPostal CHAR(5) NOT NULL,
    provincia
```

```
enum('Álava','Albacete','Alicante','Almería','Asturias','Ávila','Badajoz','Baleares','Ba

Real','Córdoba','Cuenca','Gerona','Granada','Guadalajara','Guipúzcoa','Huelva','Huesca',
 Coruña','La Rioja','Las
Palmas','León','Lérida','Lugo','Madrid','Málaga','Melilla','Murcia','Navarra','Orense','
 Cruz de
Tenerife','Segovia','Sevilla','Soria','Tarragona','Teruel','Toledo','Valencia','Vallado

    localidad VARCHAR(50) NOT NULL,
    calle VARCHAR(70), -- Hay ubicaciones que es posible que no estén en una calle
definida, pueden estar en una localización remota o no disponemos de ella en el
momento (por lo que sea). O no es necesaria una dirección porque es un lugar conocido
y característico.
    luz_dinamica BOOL,
    aire_libre BOOL,
    equipo_sonido BOOL,
    superficie SMALLINT, -- Es razonable pensar que no hay ubicaciones con más de
32km^2, si este fuese el caso es fácilmente remediable cambiando el tipo a INTEGER.
    Check(superficie>0 and alquilerUbi>=0 and aforo>0),
    PRIMARY KEY (idUbi)
);

CREATE TABLE ASISTENTE (
    emailAsist VARCHAR(320), -- El email más largo posible es de 320 caracteres.
Fuente: RFC 5321 y RFC 3696
    tfnAsist VARCHAR(38) UNIQUE, -- El número de telefono más largo posible, contando
prefijos, +, (), - y espacios nunca excede 38 caracteres. El número será único.
    nomAsist VARCHAR(70) NOT NULL, -- Exigir el nombre del asistente es razonable
    PRIMARY KEY (emailAsist)
);

CREATE TABLE ARTISTA (
    nomArt VARCHAR(70),
    cacheArt FLOAT NOT NULL, -- Es un dato especialmente importante
    bioArt VARCHAR(1000),
    PRIMARY KEY (nomArt),
    Check (cacheArt>0)
);

CREATE TABLE EVENTO (
    idEvent SMALLINT auto_increment,
    nomEvent VARCHAR(100) NOT NULL,
    descEvent VARCHAR(1500), -- Podemos prescindir de una descripción así que no
exigimos NOT NULL
    timeEvent timestamp NOT NULL,
    durEvent tinyint NOT NULL DEFAULT 1,
    actEvent enum('Concierto','Conferencia','Fiesta','Feria'), -- Asumo que la empresa
organiza un tipo limitado de actividades
    generoEvent enum('Rock','Jazz','Pop','Variado','Ciencia','Cultura','Deporte'),
    precioEntradaEvent FLOAT NOT NULL, -- Tenemos atributos derivados que dependen de
esto y es un dato importante así que exigimos NOT NULL. No usamos NUMERIC pues lo
almacena como cadena y necesitamos que sea compatible con operaciones
```

```
    total_asistentes SMALLINT DEFAULT 0,
    coste_artistas FLOAT DEFAULT 0,
    estrellasEvent NUMERIC(3,2) DEFAULT NULL,
    cod_Ubi SMALLINT NOT NULL,
    PRIMARY KEY (idEvent),
    FOREIGN KEY (cod_ubi) REFERENCES UBICACION(idUbi)
);
-- Añadimos las restricciones ahora para ilustrar como se haría
alter table EVENTO add constraint check(total_asistentes>=0);
alter table EVENTO add constraint check(durEvent>=1);
alter table EVENTO add constraint check(precioEntradaEvent>=0);

CREATE TABLE tfnArt (
    NombreArtista VARCHAR(70),
    tfnArtista VARCHAR(38) UNIQUE,
    tfnManager VARCHAR(38),
    PRIMARY KEY (NombreArtista),
    FOREIGN KEY (NombreArtista) REFERENCES ARTISTA(nomArt)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE Participa (
    id_Artista VARCHAR(70),
    id_Evento SMALLINT,
    PRIMARY KEY (id_Artista, id_Evento),
    FOREIGN KEY (id_Artista) REFERENCES ARTISTA(nomArt)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (id_Evento) REFERENCES EVENTO(idEvent)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

CREATE TABLE Atiende (
    id_Asist VARCHAR(320),
    id_AtiendeEvento SMALLINT,
    opinion TINYINT,
    PRIMARY KEY (id_Asist,id_AtiendeEvento),
    Check(opinion>=1 and opinion<=5),
    FOREIGN KEY (id_Asist) REFERENCES ASISTENTE(emailAsist),
    FOREIGN KEY (id_AtiendeEvento) REFERENCES EVENTO(idEvent)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

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

```
-- Vamos a añadir dos triggers que mantendrá el atributo total_asistentes actualizado
drop trigger if exists sumarAsistentesEvento;
drop trigger if exists restarAsistentesEvento;
drop trigger if exists actualizarCosteArtistas1;
drop trigger if exists actualizarCosteArtistas2;
drop trigger if exists calcularEstrellasEvento1;
drop trigger if exists calcularEstrellasEvento2;
drop trigger if exists calcularEstrellasEvento3;
drop trigger if exists VerificarMinimoArtistas;
drop trigger if exists verificarAforoAntesDeInsertar; -- Este trigger no está activo porque está entre comentarios justo abajo

CREATE TRIGGER sumarAsistentesEvento -- Cuando añadimos un asistente se actualiza el atributo total_asistentes
AFTER INSERT ON Atiende
FOR EACH ROW
UPDATE EVENTO
SET total_asistentes = total_asistentes + 1
WHERE idEvent = NEW.id_AtiendeEvento;

CREATE TRIGGER restarAsistentesEvento -- Cuando un asistente cancela ir a un evento se actualiza el atributo total_asistentes
AFTER DELETE ON Atiende
FOR EACH ROW
UPDATE EVENTO
SET total_asistentes = total_asistentes - 1
WHERE idEvent = OLD.id_AtiendeEvento;


/*  No llegaremos a usar este trigger con nuestros datos pues no tenemos tantos pero lo dejo aquí como comentario para ilustrar cómo se haría

DELIMITER //
CREATE TRIGGER verificarAforoAntesDeInsertar
BEFORE INSERT ON Atiende FOR EACH ROW
BEGIN
    DECLARE aforoEvento SMALLINT;
    DECLARE totalAsistentesEvento SMALLINT;

    -- Obtiene el aforo del evento desde la tabla UBICACION
    SELECT UBICACION.aforo
    INTO aforoEvento
    FROM UBICACION
    WHERE UBICACION.idUbi = (SELECT EVENTO.cod_Ubi FROM EVENTO WHERE EVENTO.idEvent = NEW.id_AtiendeEvento);

    -- Obtiene el número total de asistentes del evento desde la tabla EVENTO
    SELECT EVENTO.total_asistentes
    INTO totalAsistentesEvento
    FROM EVENTO
    WHERE EVENTO.idEvent = NEW.id_AtiendeEvento;
```

```
    -- Comprueba si agregar un nuevo asistente superaría el aforo del evento
    IF (totalAsistentesEvento + 1) > aforoEvento THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No se puede superar el aforo del local';
    END IF;
END;
//
DELIMITER ;

*/

/* ----------------------------------------------*/

DELIMITER //
CREATE TRIGGER calcularCosteArtistas1 -- Cuando un artista decide participar en uno de
los eventos actualizamos coste_artistas
AFTER INSERT ON Participa FOR EACH ROW
BEGIN
    DECLARE cacheNuevo FLOAT;

    -- Obtiene el valor del atributo cacheArt del artista relacionado con el evento
    SELECT ARTISTA.cacheArt
    INTO cacheNuevo
    FROM ARTISTA
    WHERE ARTISTA.nomArt = NEW.id_Artista;

    -- Actualiza el atributo coste_artistas del evento con la suma
    UPDATE EVENTO
    SET coste_artistas = coste_artistas + cacheNuevo
    WHERE EVENTO.idEvent = NEW.id_Evento;
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER calcularCosteArtistas2 -- Cuando un artista decide abandonar uno de los
eventos actualizamos coste_artistas
AFTER DELETE ON Participa
FOR EACH ROW
BEGIN
    DECLARE deletedCache FLOAT;

    -- Obtiene el valor del atributo cacheArt del artista que abandona el evento
    SELECT ARTISTA.cacheArt
    INTO deletedCache
    FROM ARTISTA
    WHERE ARTISTA.nomArt = OLD.id_Artista;

    -- Actualiza el atributo coste_artistas del evento con la resta
    UPDATE EVENTO
    SET coste_artistas = coste_artistas - deletedCache
    WHERE EVENTO.idEvent = OLD.id_Evento;
```

```
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER VerificarMinimoArtistas -- Comprueba que se mantiene la cardinalidad
que exigimos, debe haber al menos un artista por evento, de otra forma, no tendríamos
evento que organizar
BEFORE DELETE ON Participa
FOR EACH ROW
BEGIN
    DECLARE totalArtistas int;

    -- Calcula la cantidad de artistas que participan en el evento
    SELECT COUNT(*)
    INTO totalArtistas
    FROM Participa
    WHERE Participa.id_Evento = OLD.id_Evento;

    -- Si el total en este momento es uno no podemos permitir la eliminación de este
artista
    IF totalArtistas = 1
    THEN SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Debe haber mínimo un artista que
participe en el evento';
    END IF;
END;
//
DELIMITER ;

/* ------------------------------------------------*/
delimiter //
CREATE TRIGGER calcularEstrellasEvento1 -- Cuando un asistente valore el evento
recalculamos la media de valoraciones
AFTER UPDATE ON Atiende
FOR EACH ROW
BEGIN
    DECLARE nuevaOpinion Numeric(3,2);

    SELECT avg(opinion)
    INTO nuevaOpinion
    FROM Atiende
    WHERE Atiende.id_AtiendeEvento = NEW.id_AtiendeEvento;

    UPDATE EVENTO
    SET estrellasEvent = nuevaOpinion
    WHERE idEvent = NEW.id_AtiendeEvento;
END;
//
delimiter ;

delimiter //
CREATE TRIGGER calcularEstrellasEvento2 -- Este trigger está aquí para la inserción de
```

```
datos inicial para realizar el trabajo. Posiblemente, en el caso de uso normal de la
base de datos nunca tendríamos esto, pues los asistentes solo valoran el evento tras
ir, que es para lo que sirve el trigger anterior.
AFTER INSERT ON Atiende
FOR EACH ROW
BEGIN
    DECLARE nuevaOpinion Numeric(3,2);

    SELECT avg(Atiende.opinion)
    INTO nuevaOpinion
    FROM Atiende
    WHERE Atiende.id_AtiendeEvento = NEW.id_AtiendeEvento;

    UPDATE EVENTO
    SET estrellasEvent = nuevaOpinion
    WHERE EVENTO.idEvent = NEW.id_AtiendeEvento;
END;
//
delimiter ;

delimiter //
CREATE TRIGGER calcularEstrellasEvento3 -- Nuevamente, es posible que este trigger no
llega a usarse en el caso de uso normal de la base de datos, pues si un asistente
decide no ir al evento entonces no podría ponerle puntuación. Pero como nuestros datos
iniciales vienen con puntuaciones he considerado apropiado tenerlo.
AFTER DELETE ON Atiende
FOR EACH ROW
BEGIN
    DECLARE nuevaOpinion Numeric(3,2);

    SELECT avg(Atiende.opinion)
    INTO nuevaOpinion
    FROM Atiende
    WHERE Atiende.id_AtiendeEvento = OLD.id_AtiendeEvento;

    UPDATE EVENTO
    SET estrellasEvent = nuevaOpinion
    WHERE EVENTO.idEvent = OLD.id_AtiendeEvento;
END;
//
delimiter ;


-- INSERCIÓN DE DATOS

-- DATOS DE EJEMPLO DE UBICACION
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Lugar A', 2000.50, 300, '28002', 'Madrid', 'Madrid', 'Calle Principal 1', 1,
1, 1, 400);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
luz_dinamica, aire_libre, equipo_sonido, superficie)
```

```
VALUES ('Salón B', 1200.75, 150, '08002', 'Barcelona', 'Barcelona', NULL, 0, 1, 250);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Espacio C', 1800.25, 50, '46002', 'Valencia', 'Valencia', 'Avenida Ejemplo
3', 1, 1, 1, 300);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Sala D', 900.00, 70, '41002', 'Sevilla', 'Sevilla', 'Calle Ejemplo 4', 1, 1,
0, 350);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Centro E', 1500.25, 200, '03001', 'Alicante', 'Alicante', 'Plaza Principal',
1, 0, 1, 200);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Espacio F', 1200.75, 100, '46003', 'Valencia', 'Valencia', NULL, 1, 0, 0,
100);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Lugar G', 1600.50, 250, '28003', 'Madrid', 'Madrid', 'Avenida Principal 2',
1, 1, 1, 450);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Patio H', 800.75, 50, '08003', 'Barcelona', 'Barcelona', 'Plaza Central', 0,
1, 0, 150);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Sala I', 2000.25, 100, '46004', 'Valencia', 'Valencia', 'Calle Ejemplo 5', 1,
1, 1, 300);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Centro J', 1500.00, 200, '41003', 'Sevilla', 'Sevilla', 'Avenida Principal
3', 1, 0, 1, 400);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Lugar K', 1100.75, 80, '08004', 'Barcelona', 'Barcelona', 'Plaza Ejemplo 1',
0, 1, 0, 200);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Sala L', 1800.25, 70, '46005', 'Valencia', 'Valencia', 'Calle Principal 6',
1, 1, 1, 350);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Centro M', 1300.00, 150, '41004', 'Sevilla', 'Sevilla', 'Avenida Ejemplo 4',
1, 0, 1, 300);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Salón N', 900.75, 120, '08005', 'Barcelona', 'Barcelona', 'Plaza Central 2',
0, 1, 1, 250);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Espacio O', 1600.25, 40, '46006', 'Valencia', 'Valencia', 'Calle Ejemplo 7',
```

```
1, 1, 0, 200);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Lugar P', 1900.50, 300, '28004', 'Madrid', 'Madrid', 'Calle Principal 4', 1,
1, 1, 500);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Patio Q', 700.75, 60, '08006', 'Barcelona', 'Barcelona', 'Plaza Central 3',
0, 1, 1, 180);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Sala R', 2200.25, 90, '46007', 'Valencia', 'Valencia', 'Avenida Ejemplo 5',
1, 0, 0, 120);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Centro S', 1600.00, 180, '41005', 'Sevilla', 'Sevilla', 'Calle Principal 7',
1, 1, 1, 350);
INSERT INTO UBICACION (nombreUbi, alquilerUbi, aforo, codPostal, provincia, localidad,
calle, luz_dinamica, aire_libre, equipo_sonido, superficie)
VALUES ('Espacio T', 1200.75, 70, '08007', 'Barcelona', 'Barcelona', 'Plaza Ejemplo
2', 0, 1, 0, 220);

-- DATOS DE EJEMPLO DE EVENTOS
INSERT INTO EVENTO (nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent,
precioEntradaEvent, cod_Ubi)
VALUES ('Concierto de Rock', 'Un emocionante concierto de rock en vivo', '2023-10-20
19:30:00', 2, 'Concierto', 'Rock', 25.00, 1);
INSERT INTO EVENTO (nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent,
precioEntradaEvent, cod_Ubi)
VALUES ('Conferencia de Ciencia', 'Una charla fascinante sobre avances científicos',
'2023-11-15 10:00:00', 1, 'Conferencia', 'Ciencia', 10.00, 2);
INSERT INTO EVENTO (nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent,
precioEntradaEvent, cod_Ubi)
VALUES ('Fiesta de Verano', 'Una fiesta al aire libre con música y diversión', '2023-
07-05 21:00:00', 5, 'Fiesta', 'Variado', 15.00, 3);
INSERT INTO EVENTO (nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent,
precioEntradaEvent, cod_Ubi)
VALUES ('Concierto de Jazz', 'Un espectáculo de jazz en un entorno íntimo', '2023-09-
10 20:00:00', 2, 'Concierto', 'Jazz', 20.00, 3);
INSERT INTO EVENTO (nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent,
precioEntradaEvent, cod_Ubi)
VALUES ('Feria de Arte', 'Exposición de arte contemporáneo', '2023-08-25 11:00:00', 1,
'Feria', 'Cultura', 5.00, 5);
INSERT INTO EVENTO (nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent,
precioEntradaEvent, cod_Ubi)
VALUES ('Conferencia de Liderazgo', 'Aprende habilidades de liderazgo efectivas',
'2023-10-30 14:30:00', 1, 'Conferencia', 'Ciencia', 15.00, 6);
INSERT INTO EVENTO (nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent,
precioEntradaEvent, cod_Ubi)
VALUES ('Concierto de Pop', 'Exitosos éxitos pop interpretados en vivo', '2023-11-05
18:00:00', 2, 'Concierto', 'Pop', 30.00, 15);
INSERT INTO EVENTO (nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent,
```

```
precioEntradaEvent, cod_Ubi)
VALUES ('Deporte Extremo', 'Competiciones de deportes extremos en un entorno
emocionante', '2023-09-15 09:00:00', 8, 'Feria', 'Deporte', 40.00, 15);
INSERT INTO EVENTO (nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent,
precioEntradaEvent, cod_Ubi)
VALUES ('Fiesta de Halloween', 'Celebración espeluznante de Halloween', '2023-10-31
20:00:00', 6, 'Fiesta', 'Variado', 10.00, 9);
INSERT INTO EVENTO (nomEvent, descEvent, timeEvent, durEvent, actEvent, generoEvent,
precioEntradaEvent, cod_Ubi)
VALUES ('Conferencia de Tecnología', 'Últimas tendencias tecnológicas y desarrollos',
'2023-12-05 13:30:00', 1, 'Conferencia', 'Ciencia', 25.00, 10);

-- DATOS DE EJEMPLO DE ARTISTAS
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Artista1', 2500.50, 'Cantante famoso por sus éxitos en el pop y el rock.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Artista2', 1800.75, 'Banda de música electrónica que ha revolucionado el
género.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Artista3', 3000.25, 'Innovador músico de jazz conocido por su virtuosismo en
el saxofón.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Artista4', 1500.00, 'Cantante de country ganador de múltiples premios
Grammy.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Conferencista1', 2200.75, 'Conferencista motivacional que ha inspirado a
miles de personas.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Conferencista2', 1200.25, 'Escritor y orador conocido por sus charlas sobre
liderazgo y superación personal.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Artista5', 2700.50, 'Banda de metal pesado con una base de seguidores
leales.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Artista6', 1900.75, 'Grupo de música latina que ha alcanzado la fama
internacional.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Artista7', 3500.25, 'Compositor clásico conocido por sus sinfonías épicas.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Conferencista3', 2800.00, 'Conferencista experto en finanzas personales y
planificación del futuro.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Artista8', 1500.50, 'Banda de rock alternativo conocida por sus enérgicos
conciertos.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Conferencista4', 2200.75, 'Conferencista de tecnología y emprendimiento que
ha fundado varias startups exitosas.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Artista9', 1200.25, 'Cantante de ópera aclamado por su voz poderosa.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Artista10', 1600.00, 'Grupo de música folk que preserva la tradición musical
de su región.');
```

```
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Conferencista5', 1900.75, 'Conferencista de psicología y bienestar emocional
que ayuda a las personas a encontrar la felicidad.');
INSERT INTO ARTISTA (nomArt, cacheArt, bioArt)
VALUES ('Conferencista6', 1400.25, 'Conferencista de marketing digital que ha
asesorado a grandes empresas en estrategias de publicidad en línea.');

-- DATOS DE EJEMPLO DE ASISTENTES
INSERT INTO ASISTENTE (emailAsist, tfnAsist, nomAsist)
VALUES ('asistente1@example.com', '123-456-7890', 'Juan Pérez');
INSERT INTO ASISTENTE (emailAsist, tfnAsist, nomAsist)
VALUES ('asistente2@example.com', '987-654-3210', 'María González');
INSERT INTO ASISTENTE (emailAsist, tfnAsist, nomAsist)
VALUES ('asistente3@example.com', '555-123-4567', 'Luis Rodríguez');
INSERT INTO ASISTENTE (emailAsist, tfnAsist, nomAsist)
VALUES ('asistente4@example.com', '777-888-9999', 'Ana Martínez');
INSERT INTO ASISTENTE (emailAsist, tfnAsist, nomAsist)
VALUES ('asistente5@example.com', '333-222-1111', 'Carlos Sánchez');
INSERT INTO ASISTENTE (emailAsist, tfnAsist, nomAsist)
VALUES ('asistente6@example.com', '111-222-3333', 'Laura López');
INSERT INTO ASISTENTE (emailAsist, tfnAsist, nomAsist)
VALUES ('asistente7@example.com', '444-555-6666', 'Pedro Ramírez');
INSERT INTO ASISTENTE (emailAsist, tfnAsist, nomAsist)
VALUES ('asistente8@example.com', '222-999-8888', 'Elena Fernández');
INSERT INTO ASISTENTE (emailAsist, tfnAsist, nomAsist)
VALUES ('asistente9@example.com', '666-777-5555', 'Miguel García');
INSERT INTO ASISTENTE (emailAsist, tfnAsist, nomAsist)
VALUES ('asistente10@example.com', '888-111-4444', 'Carmen Torres');

-- DATOS DE EJEMPLO tfnArt
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Artista1', '+1234567891', '+9876543211');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Artista2', '+1234567892', '+9876543212');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Artista3', '+1234567893', '+9876543213');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Artista4', '+1234567894', '+9876543214');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Artista5', '+1234567895', '+9876543215');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Artista6', '+1234567896', '+9876543216');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Artista7', '+1234567897', '+9876543217');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Artista8', '+1234567898', '+9876543218');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Artista9', '+1234567899', '+9876543219');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Artista10', '+1234567890', '+9876543210');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Conferencista1', '+1234567881', '+9876543281');
```

```
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Conferencista2', '+1234567882', '+9876543282');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Conferencista3', '+1234567883', '+9876543283');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Conferencista4', '+1234567884', '+9876543284');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Conferencista5', '+1234567885', '+9876543285');
INSERT INTO tfnArt (NombreArtista, tfnArtista, tfnManager)
VALUES ('Conferencista6', '+1234567886', '+9876543286');

-- EJEMPLO RELACIONES ARTISTA EVENTO
-- Los datos son algo raros porque he relacionado artistas con conferencias, que no
tendría mucho sentido, pero el punto es dar datos que nos sirvan de ejemplo para sacar
consultas, en el caso de uso real los datos serían obviamente más realistas
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista1', 1);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista2', 2);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista3', 3);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista4', 4);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista5', 5);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista6', 6);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista7', 7);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista8', 8);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista9', 9);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista10', 10);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista3', 5);
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista2', 4);

-- EJEMPLO RELACION Asiste
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente1@example.com', 1, 4);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente2@example.com', 1, 5);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente3@example.com', 1, 1);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente4@example.com', 2, 4);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente5@example.com', 2, 1);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
```

```
VALUES ('asistente6@example.com', 3, 5);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente7@example.com', 3, 3);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente8@example.com', 4, 4);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente9@example.com', 4, 5);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente10@example.com', 5, 2);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente1@example.com', 6, 4);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente2@example.com', 6, 3);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente3@example.com', 7, 5);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente4@example.com', 7, 4);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente5@example.com', 8, 2);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente6@example.com', 8, 3);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente7@example.com', 9, 4);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente8@example.com', 9, 5);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente9@example.com', 10, 4);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente10@example.com', 10, 5);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente1@example.com', 5, 3);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente2@example.com', 4, 2);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion)
VALUES ('asistente3@example.com', 10, 4);
INSERT INTO Atiende (id_Asist, id_AtiendeEvento)
VALUES ('asistente4@example.com', 8);


/*--------------------------------------------------------------------------------
------------------
Consultas, modificaciones, borrados y vistas con enunciado
--------------------------------------------------------------------------------
-----------------*/

-- Veamos cómo es la tabla EVENTO primero
SELECT * FROM EVENTO;

-- Comprobamos cómo funcionan algunos triggers
-- sumarAsistentesEvento, restarAsistentesEvento, calcularEstrellasEvento1,
calcularEstrellasEvento2, calcularEstrellasEvento3.
INSERT INTO Atiende (id_Asist, id_AtiendeEvento, opinion) VALUES
```

```
('asistente3@example.com', 9, 4);
SELECT * FROM EVENTO; -- Vemos cómo el total de asistentes cambia y las estrellas
también.
DELETE FROM Atiende WHERE (id_Asist = 'asistente3@example.com' and id_AtiendeEvento =
9);
SELECT * FROM EVENTO; -- Vemos cómo el total de asistentes cambia de nuevo a los datos
iniciales y las estrellas también.
UPDATE Atiende SET opinion = 3 WHERE (id_Asist = 'asistente4@example.com' and
id_AtiendeEvento = 8);
SELECT * FROM EVENTO; -- Vemos cómo el asistente4 al añadir su puntuación del evento
se actualiza el atributo estrellasEvento apropiadamente.
UPDATE Atiende SET opinion = NULL WHERE (id_Asist = 'asistente4@example.com' and
id_AtiendeEvento = 8); -- Restauramos el valor a tal como lo teníamos al principio
SELECT * FROM EVENTO; -- Tenemos la tabla como al principio
-- actualizarCosteArtistas1, actualizarCosteArtistas2, VerificarMinimoArtistas
SELECT * FROM Participa; -- Veamos cómo es la relación entre los artistas y los
eventos. Vemos que en el evento 1 hay un único artista, vamos a probar los triggers
con este evento.
SELECT * FROM EVENTO WHERE (idEvent=1); -- Vemos los datos del evento
DELETE FROM Participa WHERE (id_Artista = 'Artista1' and id_Evento = 1); -- El trigger
VerificarMinimoArtistas evita el borrado, manteniendo así la cardinalidad que buscamos
INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista4', 1); -- Añadimos un artista para probar actualizarCosteArtistas1
SELECT * FROM EVENTO WHERE (idEvent=1); -- Vemos que se ha sumado el cache del
artista4 a coste_artistas
DELETE FROM Participa WHERE (id_Artista = 'Artista1' and id_Evento = 1); -- Si ahora
borramos este registro tenemos que coste_artistas baja y cuenta solo el cache de
artista4
SELECT * FROM EVENTO WHERE (idEvent=1); -- Efectivamente solo cuenta el cache del
artista4, que es de 1500

INSERT INTO Participa (id_Artista, id_Evento)
VALUES ('Artista1', 1); -- Volvemos a añadir el artista1 al evento 1 para tenerlo como
al principio



-- Ahora que hemos comprobado que todo funciona adecuadamente vamos a hacer algunas
consultas en nuestra base

-- Listado de los eventos con tan solo los datos relativos a los asistentes, sus
valoraciones y los artistas, sin importar datos cualitativos como la localidad o el
nombre del evento.
SELECT idEvent, precioEntradaEvent, total_asistentes, coste_artistas, estrellasEvent
FROM EVENTO;



-- ¿Cómo han sido las estadísticas de la empresa cuando ha organizado eventos de
ciencia?
SELECT idEvent, precioEntradaEvent, total_asistentes, coste_artistas, estrellasEvent
FROM EVENTO
WHERE (generoEvent LIKE 'Ciencia');
```

```
-- ¿Qué tipo de eventos ha sido mejor y peor valorado de media? Podemos verlo
ordenando de forma descendente con la valoración
SELECT actEvent as tipo, avg(estrellasEvent) as valoracion
FROM EVENTO
GROUP BY actEvent
ORDER BY valoracion desc;
-- Hemos encontrado que los mejores valorados son las fiestas y los peores son las
ferias


-- ¿Cuál es el evento más antiguo que hemos organizado, cuándo y hace cuánto fue
organizado?
SELECT nomEvent, timeEvent, timestampdiff(DAY,NOW(),timeEvent) as timeDiff
FROM EVENTO
ORDER BY timeDiff asc;
-- Vemos que el evento más antiguo que tenemos en la base de datos fue organizado el
2023-07-05, hace 98 días (a 12/10/2023)


-- ¿Cuánto debemos pagar a cada artista? Tener en cuenta que a los artistas les
pagamos con 10 días de antelación y por los eventos en los que ya han participado.
-- Primero vamos a ver qué eventos cumplen esos requisitos
SELECT idEvent, nomEvent
FROM EVENTO
WHERE (timestampdiff(DAY,NOW(),timeEvent) <= 10); -- Vemos que no se cumple para todos
los eventos
-- Podemos hacer esta consulta con inner join
SELECT nomArt AS NombreArtista, SUM(cacheArt) as DineroTotal
FROM ARTISTA
INNER JOIN Participa ON ARTISTA.nomArt = Participa.id_Artista -- Discriminamos
escogiendo a los artistas que  participan en algún evento
INNER JOIN EVENTO ON Participa.id_Evento = EVENTO.idEvent -- Discriminamos escogiendo
los id de los eventos que intersecan con el inner join anterior
WHERE timestampdiff(DAY,NOW(),timeEvent) <= 10 -- Comprobamos que el evento entra en
los parámetros.
GROUP BY ARTISTA.nomArt; -- Agrupamos por la clave primaria de ARTISTA

-- Retocando la condición WHERE podríamos cambiar las condiciones de a quién debemos
dinero, por ejemplo, imaginemos que ya pagamos a la gente de los eventos que hemos
organizado
-- con anterioridad y solo necesitamos pagar a los artistas que participarán en
nuestros eventos en los próximos 10 días u hoy mismo
SELECT nomArt AS NombreArtista, SUM(cacheArt) as DineroTotal
FROM ARTISTA
INNER JOIN Participa ON ARTISTA.nomArt = Participa.id_Artista -- Discriminamos
escogiendo a los artistas que  participan en algún evento
INNER JOIN EVENTO ON Participa.id_Evento = EVENTO.idEvent -- Discriminamos escogiendo
los id de los eventos que intersecan con el inner join anterior
WHERE timestampdiff(DAY,NOW(),timeEvent) <= 10 AND timestampdiff(DAY,NOW(),timeEvent)
>= 0 -- Comprobamos que el evento entra en los parámetros.
GROUP BY ARTISTA.nomArt; -- Agrupamos por la clave primaria de ARTISTA
```

```
-- Podemos hacer otra consulta relacionada con esta para mostrar cómo se podría además
vincular esto con la tabla de los telefonos de los artistas, para así tener
-- en el mismo listado la forma de contacto con su manager o el mismo artista para
organizar el pago.
SELECT nomArt AS NombreArtista, SUM(cacheArt) as DineroTotal, tfnArtista, tfnManager
FROM ARTISTA
LEFT JOIN tfnArt ON ARTISTA.nomArt = tfnArt.NombreArtista -- Utilizamos LEFT JOIN para
mantener todos los datos de ARTISTA, simplemente queremos añadir los teléfonos
INNER JOIN Participa ON ARTISTA.nomArt = Participa.id_Artista -- Discriminamos
escogiendo a los artistas que  participan en algún evento
INNER JOIN EVENTO ON Participa.id_Evento = EVENTO.idEvent -- Discriminamos escogiendo
los id de los eventos que intersecan con el inner join anterior
WHERE timestampdiff(DAY,NOW(),timeEvent) <= 10 AND timestampdiff(DAY,NOW(),timeEvent)
>= 0 -- Comprobamos que el evento entra en los parámetros.
GROUP BY ARTISTA.nomArt; -- Agrupamos por la clave primaria de ARTISTA


-- ¿Cuánto debemos pagar a cada ubicación? Debemos pagar las ubicaciones donde ya
hemos organizado eventos y también pagaremos por adelantado a todos los locales donde
tenemos eventos organizados en el futuro.
SELECT idUbi, nombreUbi, SUM(alquilerUbi * durEvent) as DineroTotal
FROM UBICACION
INNER JOIN EVENTO ON UBICACION.idUbi = EVENTO.cod_Ubi
GROUP BY UBICACION.idUbi;
-- Podemos ver el alquiler de cada ubicación y los días que las hemos alquilado en
total también. Así, podemos comprobrar qué ubicaciones son más caras y cuánto hemos
usado/usaremos cada ubicación.
SELECT idUbi, nombreUbi, SUM(alquilerUbi * durEvent) as DineroTotal, alquilerUbi,
SUM(durEvent) as diasAlquilado
FROM UBICACION
INNER JOIN EVENTO ON UBICACION.idUbi = EVENTO.cod_Ubi
GROUP BY UBICACION.idUbi;


-- ¿Cuánto hemos ganado o perdido con cada evento? Haremos esta consulta con una
vista. Notar que vamos a perder dinero porque nuestra base de datos tiene muy pocos
asistentes, si tuviesemos tantos asistentes registrados como una empresa real
tendríamos algunos
-- eventos con beneficios y otros con pérdidas.
CREATE VIEW beneficiosporEvento (Codigo_Evento, Nombre_Evento, Balance) as
SELECT idEvent, nomEvent, SUM(precioEntradaEvent*total_asistentes - (coste_artistas+
(alquilerUbi*durEvent)))
FROM EVENTO INNER JOIN UBICACION ON EVENTO.cod_Ubi = UBICACION.idUbi
GROUP BY idEvent;
SELECT * FROM beneficiosporEvento;


-- ¿A cuántos eventos ha asistido cada asistente? Hacemos esta consulta para usar LEFT
JOIN y ver cómo se implementa.
SELECT ASISTENTE.emailAsist as EmailAsistente, COUNT(Atiende.id_AtiendeEvento) as
EventosAsistidos
```

```
FROM ASISTENTE
LEFT JOIN Atiende ON ASISTENTE.emailAsist = Atiende.id_Asist
GROUP BY ASISTENTE.emailAsist;


-- Clientes que más han pagado en entradas
SELECT ASISTENTE.emailAsist as EmailCliente,
       SUM(EVENTO.precioEntradaEvent) as TotalPagado
FROM ASISTENTE INNER JOIN Atiende ON ASISTENTE.emailAsist = Atiende.id_Asist
INNER JOIN EVENTO ON Atiende.id_AtiendeEvento = EVENTO.idEvent
GROUP BY ASISTENTE.emailAsist
ORDER BY TotalPagado DESC;
```