Bases de datos SQL

Hugo Gómez Sabucedo
hugogomezsabucedo@gmail.com

Máster Big Data, Data Science & Inteligencia Artificial

Curso 2024-2025

Universidad Complutense de Madrid

1. Enunciado del problema

El enunciado del problema 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.

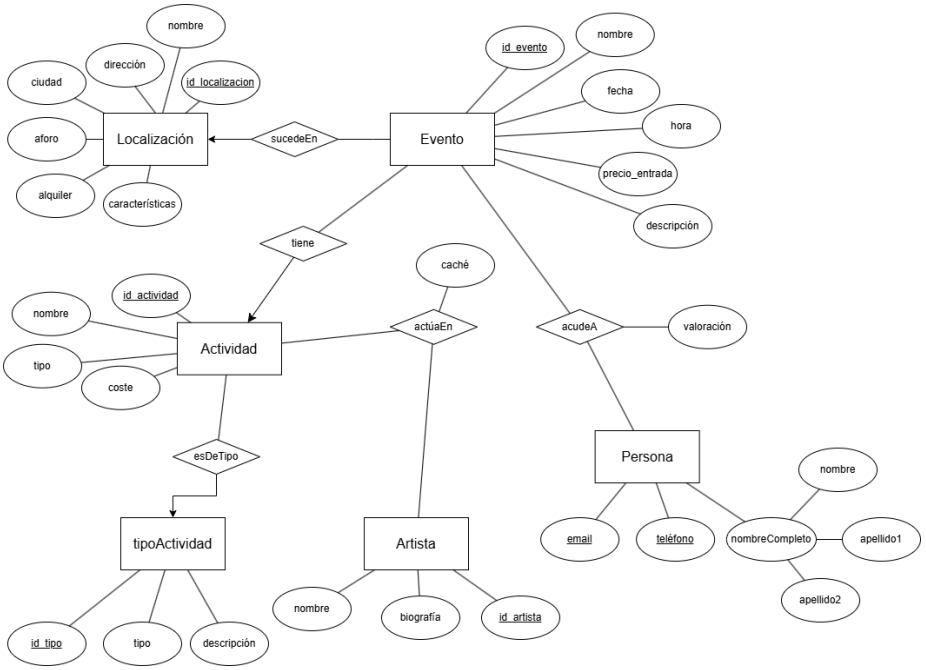
De cada evento tenemos que saber el nombre del evento (p.e. "VI festival de música clásica de Alcobendas"), la actividad, la ubicación, el precio de la entrada, la fecha y la hora, así como una breve descripción del mismo. En un evento sólo se realiza una actividad.

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, ...

2. Diseño conceptual

El diagrama entidad-relación que se propone es el siguiente:



A continuación se explica más en profundidad dicho diagrama y las diferentes entidades y relaciones que lo componen.

En resumen, se han definido seis entidades (actividad, tipoActividad, artista, evento, localización y persona), cada una de las cuales se corresponde con los principales conceptos que se mencionan en el enunciado del ejercicio. Además, se han definido cinco relaciones (sucedeEn, tiene, actúaEn, acudeA y esDeTipo) entre las entidades anteriormente mencionadas.

En primer lugar tenemos la entidad **Evento**, que representa cada uno de los eventos que realiza la empresa. Esta entidad tendrá como atributos el nombre, la fecha y hora (que en este punto he considerado como dos atributos diferenciados, aunque a posteriori en el paso a tablas pueda definirse como un único atributo, por ejemplo, de tipo *timestamp*), el precio de la entrada y la descripción del mismo. Además, se ha añadido un atributo adicional, id_evento, el cual se usará como clave primaria de la entidad. Aunque incialmente había pensado en utilizar como clave del evento su nombre y la fecha y hora, no me parece la mejor decisión, también en terminos de rendimiento.

Un evento tiene lugar en una ubicación en concreto. Es por eso que se define la entidad **Localización**, que representa las diferentes salas o lugares donde

puede tener lugar un evento. Como se indica, sus atributos son su nombre, la dirección, la ciudad, el aforo, el precio del alquiler y las características del mismo. A mayores, se ha definido un atributo adicional id_localizacion, el cual, al igual que en el caso anterior, usaremos como clave primaria de la entidad.

Entre estas dos entidades surge la relación **sucedeEn**, que es de 1 a muchos entre Evento y Localización, ya que un evento sólamente puede tener lugar en una localización, pero una misma localización puede albergar varios eventos a lo largo del tiempo.

Por una parte, se define la entidad **Persona**, que representa los diferentes asistentes a los eventos. De éstas, únicamente se nos indica que nos interesan su nombre completo, su email y su teléfono. El nombre completo lo hemos definido como un atributo compuesto, el cual dividimos a su vez en nombre, apellido1 y apellido2. Por su parte, como clave primaria se ha considerado el email y el teléfono de la persona. En el enunciado del problema no se especifica que se vayan a recoger identificadores únicos de las personas, como podría ser el DNI, por lo que este no se puede añadir como atributo. Además, emplear un identificador autogenerado y autoincremental me parecía redundante respecto al resto de entidades, y poco eficiente. Por eso he considerado el email y teléfono como la clave primaria, ya que me parece que es una combinación que, en cualquier caso, debería ser única (no tendría mucho sentido que una persona se registrase varias veces pero con diferentes combinaciones de email y teléfono).

Esta entidad tiene únicamente una relación, **acudeA**, que la relaciona con el evento. Esta relación es de muchos a muchos, ya que se indica que una persona puede asistir a varios eventos, y que a un evento asisten varias personas. Además, en el enunciado se comenta que para los eventos se desea conocer .el entusiasmo de los visitantes que asisten". Es por ello que se ha definido un atributo de esta relación, valoración, que nos permitirá registrar las opiniones de los asistentes a los diferentes eventos.

Por otra parte, se define la entidad **Actividad**, que se corresponde con cada una de las diferentes actividades que tienen lugar en el evento. Como se indica, de una actividad nos interesa su nombre, su tipo y su coste. Adicionalmente, se define el atributo id_actividad, para establecerlo como clave primaria de la entidad. Respecto al coste de la actividad, se indica que el coste es la suma del caché de los artistas. Este coste, en el modelo entidad relación, se ha definido como un atributo diferenciado, pero es importante destacar que depende de la relación entre actividad y artista (un artista, por ejemplo, puede tener un caché diferente para diferentes actividades). En el paso a tablas se explicará como se abordará esta cuestión en nuestra base de datos. La actividad se relaciona con evento mediante la relación **tiene**, de 1 a muchos entre evento y actividad. Es decir, un evento sólo tiene una actividad, pero una actividad puede estar en varios eventos.

Respecto a la actividad, se indica que puede ser de diferentes tipos. A priori, podría parecer lógico definir el tipo de la actividad como un atributo de la

propia entidad. Sin embargo, parece más adecuado definir el tipo de actividad como una entidad diferenciada, ya que los tipos de actividad serán siempre los mismo (dentro de un conjunto), y esto nos permite además definir atributos adicionales como por ejemplo una breve descripción. Es por esto que nace la entidad **tipoActividad**, cuyos atributos son un id_tipo, así como el nombre o tipo y la descripción. Esta entidad se relaciona con Actividad mediante la relación **esDeTipo**, que es de cardinalidad varios a uno. Es decir, una actividad sólamente es de un tipo en concreto, pero un tipo puede tener (y tiene) varias actividades.

Por último, se define la entidad **Artista**, que únicamente tiene como atributos el nombre del mismo y su biografía, así como un atributo adicional id_artista que se usará como clave primaria. El Artista se relaciona con la Actividad (ya que se indica en el enunciado que un artista participa en una actividad, y no en un evento), mediante la relación **actúaEn**. Esta relación tiene un atributo, caché, que representa el caché del artista. Como se comentó anteriormente, el caché de un artista no es una cantidad fija, sino que dependiendo de la actividad, un artista puede tener un caché diferente. Por eso, no puede considerarse un atribto de la entidad artista, ya que no es algo intrínseco a él, sino que se establece al crearse la actividad en cuestión. Es por eso que se ha decidido incluir como un atributo de la relación actúaEn. Además, esta relación es de muchos a muchos: se indica que en cada actividad participan uno o varios artistas y, naturalmente, un artista puede participar en múltiples actividades.

3. Diseño lógico

Una vez que tenemos definidas todas las entidades, podemos realizar el diseño lógico de la base de datos, mediante el paso a tablas del modelo entidad relación para obtener el modelo relacional.

En primer lugar, deberemos crear una tabla por cada una de las entidades que tenemos en el MER. Esto nos dará lugar a seis tablas, con sus correspondientes atributos: evento, localización, persona, actividad, tipoActividad y artista.

Por otra parte, aquellas relaciones que son de muchos a muchos generarán una tabla. De esta forma, se añaden dos nuevas tablas: acudeA y actúaEn. En cada una de estas tablas, guardaremos las claves primarias de cada una de las entidades participantes, así como los atributos, si los hubiese, de dicha relación. Para las que son de 1 a muchos, se pasará la clave primaria de la entidad con participación 1 a la entidad con participación n.

Como detalle, destacar que los atributos *fecha* y *hora* de la entidad Evento se han fusionado en uno solo, fecha, ya que a nivel de implementación en SQL se puede definir como un tipo timestamp, facilitando el trabajo con los datos.

De esta forma, el modelo relacional es el siguiente:

LOCALIZACION(id_localizacion, nombre, direccion, ciudad, aforo, alquiler, caracteristicas)

EVENTO(id_evento, *id_localizacion*, fecha, precio_entrada, descripción)

ACTIVIDAD(id_actividad, *id_evento*, *id_tipo*, nombre, coste)

TIPOACTIVIDAD(id_tipo, tipo, descripcion)

ARTISTA(id_artista, nombre, biografia)

PERSONA(email, telefono, nombre, apellido1, apellido2)

ACTUAEN(*id_artista*, *id_actividad*, cache)

ACUDEA(*email*, *telefono*, *id_evento*, valoracion)

4. Implementación y consultas

El script completo de la implementación de la base de datos se encuenta en el Anexo **??**. Sin embargo, aquí se explicará la elección de algunos tipos de los datos, así como un breve detalle de los triggers y vistas definidos y de las consultas propuestas, aclarando brevemente su objetivo y como se llega al mismo.

Lo primero que debemos hacer es comprobar que la base de datos cumple las tres formas normales:

- Respecto a la Primera Forma Normal (**FN1**), esta establece que no debe haber grupos repetidos de columnas ni una columna con múltiples valores. Esto se comprueba viendo que todas las tablas tienen atributos atómicos y una clave primaria que asegura la unicidad de los registros.
- La Segunda Forma Normal (**FN2**) dice que, además de cumplirse la FN1, debe haber dependencia funcional: los atributos que no forman parte de ninguna clave deben depender completamente de la clave primaria. Esto se cumple viendo que todas las tablas con claves compuestas (`ActuaEn` y `AcudeA`) sus atributos dependen de toda la clave (en caso de ActuaEn, cache depende del id_artista e id_actividad; y en el caso de AcudeA, valoracion depende de email, telefono e id_evento); y para el resto, las claves primarias son simples
- Por último, la Tercera Forma Normal (**FN3**) establece que, además de cumplirse la FN2, no debe haber ninguna dependencia transitiva; es decir, cada columna debe estar relacionada directamente con las columnas de la clave primaria. Esto se comprueba directamente, por ejemplo, en la tabla Persona, donde vemos que nombre, apellido1 y apellido2 dependen directamente de la clave compuesta (email, telefono).

Una vez comprobamos las formas normales, pasamos a explicar brevemente algunos detalles sobre la implementación en SQL de la base de datos. En general, se han elegido los tipos de datos en función del atributo y su naturaleza, siendo por ejemplo la mayor parte de las claves números enteros, puesto que las definimos como un id correlativo. Valores como el alquiler de una localización, el precio de una entrada de un evento o el coste de una actividad se han definido como `DECIMAL`, con un total de 10 dígitos de los cuales 2 son decimales. Para los valores de tipo `varchar`, sus longitudes se han determinado en función del posible valor que almacenarán; por su parte, valores como la biografía del artista o las caracaterísticas de una localización se han definido como tipo `TEXT` para que se pueda almacenar una mayor cantidad de información en los mismos.

Respecto a las tablas, para casi todas aquellas claves foráneas se ha definido una política de borrado y actualización en cascada, para que al eliminar algún registro de una tabla, se elimine también de la tabla dependiente. Únicamente

en el caso de la tabla Actividad se ha definido que, en caso de que se elimine un tipo de actividad, ese campo se ponga a null en la tabla de Actividad, ya que creo que tiene más sentido en este caso guardar un valor nulo.

En lo que respecta a los triggers, se han creado dos, uno de los cuales está duplicado para que se ejecute tanto tras un update como tras un insert: el primero de ellos nos servirá para actualizar el coste de una actividad. Este valor, por defecto, lo hemos establecido a 0 (se supone que, cuando definamos inicialmente una actividad, quizás no sabremos aún los artistas). Pero, como indicamos, el coste depende del caché de los artistas, y este puede o bien variar (ya que un artista puede aumentar o disminuir su caché puntualmente) o simplemente aumentar debido a que se sumen más artistas. Por tanto, se han creado dos triggers idénticos (`actualiza_coste_actividad_insert` y `actualiza_coste_actividad_update`), que se ejecutarán tras cada insert o update en la tabla ActuaEn, que actualizarán el coste de la actividad correspondiente, estableciéndolo como la suma de los cachés de las actividades que coincidan, empleando un coalesce(caché, 0) para evitar posibles errores.

Por otra parte, tenemos el trigger `valida_aforo`, el cual nos sirve para comprobar que no se exceda el aforo de un recinto. En este caso, se ejecuta tras cada insert en AcudeA, y lo que hace es, por una parte, obtener el aforo máximo de la localización donde tiene lugar el evento al que acude la persona en cuestión; y por otra, obtener el aforo actual o número de personas que tiene ese evento. Si el aforo actual fuese mayor o igual que el máximo, no nos dejaría hacer el insert, y mostraría un mensaje por pantalla. Se muestra un ejemplo de funcionamiento de este caso, aunque comentado, para permitir que el script se ejecute correctamente por completo.

En lo que respecta a las vistas, se ha creado una sencilla, `ValoracionesEventos`, que nos permite ver, para cada evento, la valoración que le han dado los asistentes, así como el total de personas que han asistido a ese evento. Además, se ha creado la vista `GananciasEvento`, que nos permite obtener las ganancias de cada evento, teniendo en cuenta que las ganancias son la suma de todas las entradas vendidas en ese evento, restando el coste de todas las actividades y el alquiler del recinto.

Las consultas que se han propuesto son las siguientes:

1. Total de eventos por ciudad: obtener el número total de eventos en cada ciudad.
2. Sabiendo el coste promedio de las actividades, seleccionar aquellas que superen dicho coste promedio.
3. Obtener las ganancias de cada uno de los artistas en las actividades que han participado con nuestra empresa.
4. Obtener los eventos menos populares, es decir, aquellos con menor asistencia

A. Anexo: Script de SQL

```sql
/* ------------------------------------------------------
Hugo Gomez Sabucedo
ArteVida Cultural
------------------------------------------------------ */
/* ------------------------------------------------------
Definicion de la estructura de la base de datos
------------------------------------------------------
*/
DROP DATABASE IF EXISTS ArteVidaCultural;

CREATE DATABASE ArteVidaCultural;

use ArteVidaCultural;

DROP TABLE IF EXISTS Localizacion;
CREATE TABLE Localizacion(
    id_localizacion INT AUTO_INCREMENT,
    nombre VARCHAR(75),
    direccion VARCHAR(100),
    ciudad VARCHAR(50),
    aforo INT,
    alquiler DECIMAL(10,2),
    caracteristicas TEXT,

    PRIMARY KEY (id_localizacion)
);

DROP TABLE IF EXISTS Evento;
CREATE TABLE Evento(
    id_evento INT AUTO_INCREMENT,
    id_localizacion INT,
    fecha TIMESTAMP,
    precio_entrada DECIMAL (10,2),
    descripcion VARCHAR(500),

    PRIMARY KEY (id_evento),
    FOREIGN KEY (id_localizacion) REFERENCES Localizacion(
    id_localizacion)
    ON DELETE CASCADE ON UPDATE CASCADE
);

DROP TABLE IF EXISTS TipoActividad;
CREATE TABLE TipoActividad(
    id_tipo INT AUTO_INCREMENT,
    tipo VARCHAR(30),
    descripcion VARCHAR(100),

    PRIMARY KEY (id_tipo)
);
```

```
48
49 DROP TABLE IF EXISTS Actividad;
50 CREATE TABLE Actividad(
51     id_actividad INT AUTO_INCREMENT,
52     id_evento INT,
53     id_tipo INT,
54     nombre VARCHAR(50),
55     coste DECIMAL(10,2) DEFAULT 0,
56
57     PRIMARY KEY (id_actividad),
58     FOREIGN KEY (id_evento) REFERENCES Evento(id_evento)
59     ON DELETE CASCADE ON UPDATE CASCADE,
60     FOREIGN KEY (id_tipo) REFERENCES TipoActividad(id_tipo)
61     ON DELETE SET NULL ON UPDATE CASCADE
62 );
63
64 DROP TABLE IF EXISTS Artista;
65 CREATE TABLE Artista(
66     id_artista INT AUTO_INCREMENT,
67     nombre VARCHAR(50),
68     biografia TEXT,
69
70     PRIMARY KEY (id_artista)
71 );
72
73 DROP TABLE IF EXISTS Persona;
74 CREATE TABLE Persona(
75     email VARCHAR(75),
76     telefono CHAR(9),
77     nombre VARCHAR(25),
78     apellido1 VARCHAR(40),
79     apellido2 VARCHAR(40),
80
81     PRIMARY KEY (email, telefono)
82 );
83
84 DROP TABLE IF EXISTS ActuaEn;
85 CREATE TABLE ActuaEn(
86     id_artista INT,
87     id_actividad INT,
88     cache DECIMAL(10,2),
89
90     PRIMARY KEY (id_artista, id_actividad),
91     FOREIGN KEY (id_artista) REFERENCES Artista(id_artista)
92     ON UPDATE CASCADE ON DELETE CASCADE,
93     FOREIGN KEY (id_actividad) REFERENCES Actividad(
       id_actividad)
94     ON UPDATE CASCADE ON DELETE CASCADE
95 );
96
97 DROP TABLE IF EXISTS AcudeA;
```

```sql
CREATE TABLE AcudeA(
    email VARCHAR(75),
    telefono CHAR(9),
    id_evento INT,
    valoracion INT,

    PRIMARY KEY (email, telefono, id_evento),
    FOREIGN KEY (email, telefono) REFERENCES Persona(email,
    telefono)
    ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (id_evento) REFERENCES Evento(id_evento)
    ON DELETE CASCADE ON UPDATE CASCADE
);

/* ------------------------------------------------------------
Trigger
Insercion de datos
------------------------------------------------------------
*/

DROP TRIGGER IF EXISTS actualiza_coste_actividad_insert;

DELIMITER $$
CREATE TRIGGER actualiza_coste_actividad_insert
AFTER INSERT ON ActuaEn
FOR EACH ROW
BEGIN
    UPDATE Actividad
    SET coste = (
        SELECT COALESCE(SUM(cache), 0)
        FROM ActuaEn
        WHERE id_actividad = NEW.id_actividad
    )
    WHERE id_actividad = NEW.id_actividad;
END;
$$
DELIMITER ;

DROP TRIGGER IF EXISTS actualiza_coste_actividad_update;

DELIMITER $$
CREATE TRIGGER actualiza_coste_actividad_update
AFTER UPDATE ON ActuaEn
FOR EACH ROW
BEGIN
    UPDATE Actividad
    SET coste = (
        SELECT COALESCE(SUM(cache), 0)
        FROM ActuaEn
        WHERE id_actividad = NEW.id_actividad
    )
```

```
147     WHERE id_actividad = NEW.id_actividad;
148 END;
149 $$
150 DELIMITER ;
151
152 DROP TRIGGER IF EXISTS valida_aforo;
153
154 DELIMITER $$
155 CREATE TRIGGER valida_aforo
156 BEFORE INSERT ON AcudeA
157 FOR EACH ROW
158 BEGIN
159   DECLARE AforoMax INT;
160     DECLARE AforoAct INT;
161
162     SELECT l.aforo
163     INTO AforoMax
164     FROM Evento e
165     JOIN Localizacion l
166         ON e.id_localizacion = l.id_localizacion
167     WHERE e.id_evento = NEW.id_evento;
168
169     SELECT COUNT(*)
170     INTO AforoAct
171     FROM AcudeA
172     WHERE id_evento = NEW.id_evento;
173
174     IF AforoAct >= AforoMax THEN
175         SIGNAL SQLSTATE '45000'
176         SET MESSAGE_TEXT = 'No se pueden registrar mas
            asistentes que el aforo maximo del recinto.';
177     END IF;
178 END;
179 $$
180 DELIMITER ;
181
182 /* -- Insercion de datos -- */
183 INSERT INTO Localizacion (nombre, direccion, ciudad, aforo,
    alquiler, caracteristicas) VALUES
184 ('Teatro Principal', 'r a Nova 21', 'Santiago de Compostela
    ', 50, 1500.00, 'Teatro construido en 1841 que acoge
    diferentes espect culos'),
185 ('Auditorio Mar de Vigo', 'avenida da Beiramar 29', 'Vigo',
    1500, 5000.00, 'Palacio de Congresos con un gran auditorio
    situado enfrente del puerto'),
186 ('Coliseum', 'r a Francisco P rez Carballo', 'A Coru a',
    11000, 26000.00, 'Estadio de uso para conciertos y
    espect culos y partidos de baloncesto'),
187 ('Sala Capitol', 'r a Concepci n Arenal 5', 'Santiago de
    Compostela', 3, 160.00, 'Sala de conciertos (simulaci n)');
188
```

```
189 INSERT INTO Evento (id_localizacion, fecha, precio_entrada,
    descripcion) VALUES
190 (1, '2024-12-01 20:00:00', 35.00, 'Concierto de m sica
    cl sica'),
191 (2, '2024-12-10 18:00:00', 25.00, 'Conferencia sobre arte
    contempor neo'),
192 (3, '2025-06-30 21:45:00', 75.00, 'Concierto de Rosal a'),
193 (4, '2024-12-15 21:00:00', 50.00, 'Concierto de Sabela
    Rodr guez');
194
195 INSERT INTO TipoActividad (tipo, descripcion) VALUES
196 ('Concierto', 'Concierto musical de un artista'),
197 ('Conferencia', 'Charla o exposici n sobre diversos temas')
    ,
198 ('Teatro', 'Representaci n de una obra dram tica o comedia
    ');
199
200 INSERT INTO Actividad (id_evento, id_tipo, nombre) VALUES
201 (1, 1, 'Concierto de piezas de Beethoven'),
202 (1, 1, 'Concierto de piezas de Mozart'),
203 (2, 2, 'Arte en el Siglo XXI'),
204 (3, 1, 'Actuaci n musical de Sabela'),
205 (4, 1, 'Actuaci n musical de Rosal a'),
206 (4, 1, 'Actuaci n musical de los teloneros de Rosal a');
207
208 INSERT INTO Artista (nombre, biografia) VALUES
209 ('Rosal a', 'La cantante espa ola m s conocida'),
210 ('Ana Mena', 'Cantante espa ola de pop'),
211 ('Sabela Rodr guez', 'Cantante gallega con temas en gallego
    '),
212 ('Tanxugueiras', 'Tr o de cantantes gallegas'),
213 ('Juan L pez', 'Actor de teatro con una amplia trayectoria
    en obras cl sicas'),
214 ('Carlos Ruiz', 'Pianista concertista, especializado en
    m sica barroca');
215
216 INSERT INTO Persona (email, telefono, nombre, apellido1,
    apellido2) VALUES
217 ('anagomez@gmailcom', '655316631', 'Ana', 'G mez', '
    Rodr guez'),
218 ('lupesa@gmail.com', '726654546', 'Luis', 'P rez', '
    S nchez'),
219 ('carlosml@gmail.com', '634646665', 'Carlos', 'Mart n', '
    Lopez'),
220 ('hugogosa@gmail.com', '695321467', 'Hugo', 'G mez', '
    Sabucedo');
221
222 INSERT INTO AcudeA (email, telefono, id_evento, valoracion)
    VALUES
223 ('anagomez@gmailcom', '655316631', 1, 7),
224 ('anagomez@gmailcom', '655316631', 2, 8),
```

```
('hugogosa@gmail.com', '695321467', 2, 7),
('hugogosa@gmail.com', '695321467', 3, 10),
('anagomez@gmailcom', '655316631', 4, 9),
('hugogosa@gmail.com', '695321467', 4, 7),
('lupesa@gmail.com', '726654546', 4, 5);

INSERT INTO ActuaEn (id_artista, id_actividad, cache) VALUES
(1,5,1500.20),
(2,6,987.65),
(3,4,127.4),
(4,4,156.6),
(4,6,73.91),
(5,1,125.4),
(6,2,200.5);

/* ------------------------------------------------------------
Consultas, modificaciones, borrados y vistas
------------------------------------------------------------ */
set sql_safe_updates=0;

-- Comprobamos el funcionamiento del trigger
-- INSERT INTO AcudeA (email, telefono, id_evento, valoracion) VALUES ('carlosml@gmail.com', '634646665' , 4, 10);

CREATE VIEW ValoracionesEventos AS
SELECT e.id_evento, e.descripcion as evento, l.nombre as localizacion, avg(aa.valoracion) as valoracion, count(aa.email) as asistencia
FROM Evento e
JOIN Localizacion l
    ON e.id_localizacion = l.id_localizacion
LEFT JOIN AcudeA aa
    ON e.id_evento = aa.id_evento
GROUP BY e.id_evento, e.descripcion, l.nombre
;

-- Consulta 1: eventos por ciudad
SELECT l.ciudad, count(e.descripcion)
FROM Evento e
JOIN Localizacion l
  ON e.id_localizacion = l.id_localizacion
GROUP BY l.ciudad

-- Consulta
```