Universitat Oberta de Catalunya

uoc.edu

Diseño e implementación de una base de datos de eventos

Daniel Bertrand Pilot Combarro
Grado de Ingeniería Informática
Bases de Datos

Ismael Campanario Cabrera
Josep Curto Díaz

Junio 2024

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño e implementación de una base de datos de eventos*
Nombre del autor:	*Daniel Bertrand Pilot Combarro*
Nombre del consultor/a:	*Ismael Campanario Cabrera*
Nombre del PRA:	*Josep Curto Díaz*
Fecha de entrega (mm/aaaa):	06/2024
Titulación:	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Bases de Datos*
Idioma del trabajo:	*Castellano*
Palabras clave	*Gestión de eventos, base de datos relacional, PostgreSQL*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

El objetivo de este trabajo es el diseño y la implementación de una base de datos para gestión de eventos, de manera que se puedan exponer de manera pública los eventos más cercanos a un usuario y clasificarlos por categoría. La base de datos podrá posteriormente ser usada por un portal web o aplicación móvil que exponga los datos a diferentes usuarios de una región geográfica.

Tras la pandemia de COVID-19, la asistencia a eventos se ha disparado y ha vuelto a niveles previos de la pandemia, por lo que soluciones que permitan promocionar los eventos más cercanos a un usuario pueden suponer una solución que, además de prestar un servicio a la sociedad, puede resultar rentable a nivel económico.

Puesto que los requisitos del proyecto se consideran estables y poco propensos a cambios durante la duración del proyecto, se ha optado por usar el modelo en cascada para cumplir con los plazos de entrega.

Abstract (in English, 250 words or less):

The aim of this work is to design and implement an event management database which will expose and categorize the closest events to a user. This database can subsequently be used by a website or mobile application to display its data to different users in a specific area.

Following COVID-19 pandemic, event attendance has recovered and reached pre-pandemic levels. This means that developing solutions that promote events to users provides a service to society and can potentially be profitable.

Since the project requirements are considered stable and unlikely to change, we have chosen the waterfall model to meet delivery deadlines.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

En este proyecto se plantea el diseño e implementación de una base de datos que tiene como objetivo almacenar y gestionar información de una amplia gama de eventos que tengan lugar en un área geográfica específica.

Los eventos desempeñan un papel central en la vida social de las personas, proporcionando oportunidades para establecer contactos sociales, profesionales y enriquecerse culturalmente.

Sin embargo, durante la pandemia de COVID-19, se impusieron una serie de restricciones sanitarias que han impedido o dificultado ampliamente la realización de eventos presenciales. A medida que la situación ha ido mejorando, el interés de los organizadores y los asistentes se ha ido renovando y ha contribuido a la recuperación de este sector gravemente afectado.

En el contexto de postpandemia en el que nos encontramos, la asistencia a ferias y congresos ha alcanzado los niveles previos a la pandemia [1], lo que nos termina de demostrar el potencial de este sector en el futuro.

Incluso de forma previa a la pandemia, no es excepcional que una persona no se entere de los eventos que existen cerca suya que le puedan ser de interés. Teniendo en cuenta esta situación, la existencia de un sistema que permita conectar al público potencial con eventos que se encuentren en su proximidad o puedan ser de su interés es una situación deseable.

Teniendo en cuenta las características de la sociedad, y en la perspectiva de la era digital en la que vivimos actualmente, la geolocalización resulta crucial a la hora de poder ubicar los eventos en función de la cercanía con el usuario. En este caso, será además particularmente importante en el contexto de zonas más densamente pobladas, donde la concentración de eventos será mayor e incluso se pueden dar casos de varios eventos simultáneos.

Por otra parte, dado que los intereses de los usuarios son muy variables, y que los eventos tienen características, tamaños y públicos objetivos muy diferentes, es crucial poder clasificarlos correctamente para poder ofrecerle a los usuarios un contenido que pueda ser de su relevancia.

Además, la potencial implementación de funcionalidades para el usuario como pueden ser el marcar eventos como favoritos o dejar valoraciones de los diferentes eventos a los que asisten también permitirían a la empresa gestora de la plataforma conocer mejor a sus usuarios y entre ellos el aumento de la sensación de comunidad.

Planteamos, así, el caso de una empresa que pueda explotar esta base de datos para generar un portal web y una aplicación móvil que permita a los

usuarios ver los eventos cercanos y, a su vez, poder filtrarlos y guardarlos en función de las categorías y de sus gustos.

Por otra parte, la empresa podría generar ingresos adicionales comercializando entradas para ciertos eventos desde la propia plataforma, y los usuarios ya efectuarían las compras de entradas desde su misma cuenta.

Con esto concluimos que esta solución no es únicamente interesante a nivel de facilitar información para uso público, sino que además permite abrir oportunidades de negocio a empresas implementando modelos de ingresos basados en publicidad y servicios adicionales, como puede ser la venta de entradas o promoción efectiva de eventos.

Además, sabiendo que trabajamos en un entorno digital que se encuentra en cambio constante y la cantidad de usuarios y eventos listados puede variar en un periodo corto de tiempo, las soluciones desarrolladas deben permitir que la empresa se adapte a las nuevas condiciones de mercado que disponga.

1.2. Objetivos del Trabajo

El principal objetivo de este trabajo será el diseño e implementación de una base de datos de eventos que pueda ser posteriormente explotada por aplicativos para dar funcionalidad a un sistema de gestión de eventos. El desarrollo de dichos aplicativos no está incluido en este trabajo.

La solución desarrollada, además, deberá cumplir además las siguientes características:

- **Disponer de una estructura eficiente**: teniendo en cuenta las características y requisitos del proyecto, trabajaremos en una arquitectura que nos permita evitar la redundancia de datos, mantener su precisión e integridad y permitir su acceso de forma útil.

- **Optimizar el rendimiento de las consultas**: para mejorar el rendimiento de los aplicativos conectados a las bases de datos, el diseño deberá facilitar la rápida recuperación de los datos más comúnmente requeridos.

- **Facilitar la trazabilidad de las acciones realizadas**: la base de datos debe disponer de un histórico de acciones realizadas en ella, de manera que se puedan seguir los diferentes cambios realizados, depurar errores o ser usados en caso de auditoría.

- **Proveer datos analíticos**: la base de datos dispondrá de un repositorio estadístico que incluya datos de algunas de las consultas más relevantes.

- **Implementar procedimientos de alta, baja y modificación (ABM)**: para mejorar el rendimiento y la seguridad de la base de datos, se

implementarán procedimientos de alta, baja y modificación para algunas de las principales entidades de la base de datos.

- **Garantizar la seguridad de los datos**: ya que la aplicación dispone de usuarios, debe garantizarse la seguridad de sus datos. Concretamente, para los datos de los usuarios deberá cumplirse el Reglamento General de Protección de Datos (RGPD).

Por otra parte, la realización de este trabajo tiene como objetivo mostrar las habilidades y conocimientos adquiridos en el transcurso del grado y, en particular, en las asignaturas de “Gestión de Proyectos”, “Uso de bases de datos”, “Diseño de Bases de Datos” y “Arquitectura de bases de datos”.

1.3. Enfoque y método seguido

En este caso no existe un sistema previo a partir del cual partir, por lo que este trabajo consistirá en el diseño e implementación de una base de datos nueva.

En cuanto a la metodología a usar para la implementación del proyecto, decidimos que dadas las características de éste el mejor enfoque posible a adoptar es el “Modelo de cascada”, que establece una serie de fases lineales y dependientes entre si. Algunos puntos que nos han llevado a la toma de esta decisión son los siguientes:

- Los requisitos se establecen en un punto inicial y no se esperan cambios significativos de éstos a lo largo del proyecto. El modelo de cascada establece una secuencia lineal y rígida de actividades por lo que es adecuado para esta característica.

- El proyecto es desarrollado en solitario por lo que, al no haber equipo, no se esperan bloqueos entre diferentes miembros de éste ni necesidad de coordinar tareas entre los diferentes miembros.

- La realización de este trabajo está marcada por una serie de hitos y una fecha de entrega determinada, por lo que el modelo de cascada puede ser una gran ayuda para la planificación de los recursos al poder establecer plazos realistas que permitan cumplir con los objetivos del proyecto.

- El modelo de cascada es muy sencillo de implementar.

El modelo en cascada, aplicado a este proyecto, puede ser dividido en las siguientes fases:

- **Requisitos**: el primer punto será obtener un listado completo de los requisitos del proyecto. Dado que el modelo de cascada es poco flexible, esta fase es crucial para el posterior desarrollo del proyecto.

- **Diseño**: una vez se han obtenido los requisitos, podremos empezar a trabajar en el diseño de la base de datos propiamente dicho, definiendo las diferentes estructuras de datos que dispondremos (incluyendo entidades, sus atributos, las relaciones entre ellas o restricciones).

- **Implementación**: en esta fase se creará un producto funcional partiendo del diseño realizado. Algunas de las principales tareas de esta fase serán la creación de las tablas y relaciones definidas en el modelo, la implementación de restricciones y el desarrollo de los procedimientos de alta, baja y modificación necesarios.

- **Pruebas**: una vez finalizada la implementación de la base de datos, deben realizarse pruebas para comprobar su correcto funcionamiento, que el comportamiento es el esperado y que los requisitos se cumplen.

- **Entrega**: una vez se ha comprobado el correcto funcionamiento del proyecto, se presentan los entregables que posteriormente podrán ser usados.

Existe además una fase adicional de mantenimiento cuya realización se contempla una vez el producto ha sido entregado. Debido a la naturaleza de este proyecto, esta fase no estaría incluida en el alcance de este Trabajo Fin de Grado.

1.4. Planificación del Trabajo

1.4.1. Hitos y tareas del proyecto

El trabajo está definido en una serie de hitos, marcados en el tiempo, por las diferentes PEC propuestas por la UOC:

	Fecha Inicio	Fecha Fin
PEC 1	28/02/2024	11/03/2024
PEC 2	12/03/2024	15/04/2024
PEC 3	16/04/2024	20/05/2024
Entrega final	21/05/2024	17/06/2024
Tribunal de evaluación	24/06/2024	28/06/2024

Para estimar los días que llevará la realización de las diferentes tareas a realizar, debemos tener en cuenta que en mi situación personal trabajo a tiempo completo como desarrollador en una empresa. Eso implica que, entre semana, la disponibilidad será de 2 o 3 horas a final del día, y más amplia durante los fines de semana.

Debe tenerse en cuenta, además, que todas las tareas a realizar deben estar debidamente documentadas en la memoria y por tanto el tiempo de redacción se incluye en la estimación de la propia tarea.

Habiendo elegido la metodología en cascada y teniendo claras las fases a realizar, podemos definir las siguientes tareas:

Tarea	Tiempo	Fecha inicio	Fecha fin
PEC1	**20h**	**28/02/2024**	**11/03/2024**
Introducción, objetivos y planificación del trabajo	20h	28/02/2024	11/03/2024
PEC2	**60h**	**12/03/2024**	**15/04/2024**
Definición y documentación de requisitos	14h	12/03/2024	17/03/2024
Correcciones PEC1	4h	18/04/2024	20/04/2024
Diseño conceptual de la base de datos	20h	21/03/2024	28/03/2024
Diseño lógico de la base de datos	10h	29/03/2024	03/04/2024
Diseño físico de la base de datos	8h	04/04/2024	07/04/2024
Preparación del entorno de desarrollo	4h	08/04/2024	09/04/2024
PEC3	**82h**	**16/04/2024**	**20/05/2024**
Desarrollo de scripts de creación de la base de datos	30h	10/04/2024	22/04/2024
Correcciones PEC2	12h	23/04/2024	26/04/2024
Procedimientos ABM	20h	27/04/2024	06/05/2024
Creación logs	10h	07/05/2024	11/05/2024
Repositorio estadístico	10h	12/05/2024	16/05/2024
Entrega Final	**65h**	**21/05/2024**	**17/06/2024**
Plan de pruebas	10h	17/05/2024	21/05/2024
Correcciones derivadas del plan de pruebas	10h	22/05/2024	27/05/2024
Correcciones PEC3	15h	28/05/2024	02/06/2024
Preparación y grabación de presentación virtual	30h	03/06/2024	16/06/2024
Autoinforme de evaluación	2h	17/06/2024	17/06/2024
Tribunal de evaluación virtual	**10h**	**24/06/2024**	**28/06/2024**
Participación en el tribunal de evaluación virtual	10h	24/06/2024	28/06/2024

A continuación, mostramos un Diagrama de Gantt con el trabajo planificado:

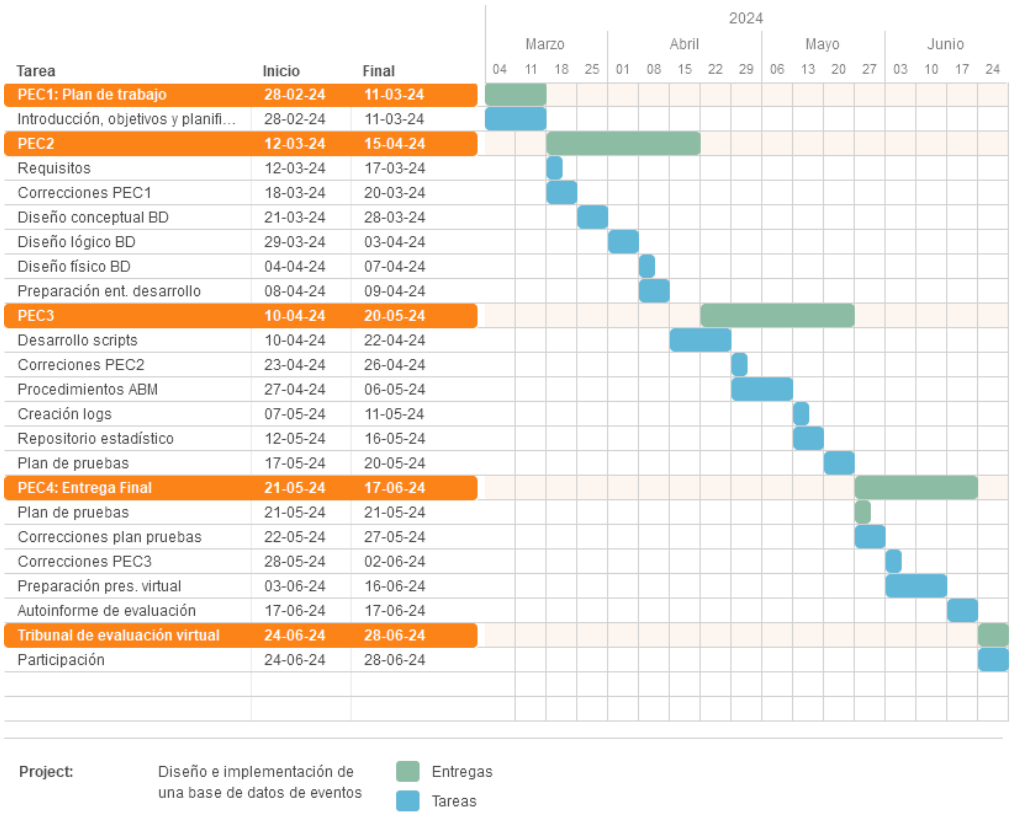


Figura 1: Diagrama de Gantt del proyecto

1.4.2. Planificación de la gestión de los riesgos

Al igual que en cualquier otro proyecto, la realización de este trabajo no está exenta de riesgos que pueden comprometer su correcta planificación y realización. Identificar, analizar y planificar la respuesta a los riesgos nos permitirá limitar el impacto que éstos podrán tener sobre nuestro proyecto.

Código	Riesgo	Consecuencia	Probabilidad	Impacto	Nivel
R01	Dedicación menor que la estimada por complicaciones laborales o personales.	Retrasos en las entregas del proyecto. Posible incumplimiento de objetivos.	Media	Alto	Alto
R02	Cambio de los requisitos del proyecto	Necesidad de reajustar el diseño y desarrollo.	Baja	Alto	Medio
R03	Pérdida de datos por fallos del sistema	Interrupción del proyecto, necesidad de volver a hacer partes ya hechas, retrasos.	Baja	Alto	Medio
R04	Error de estimación de las tareas y planificación	Incumplimiento de plazos de entregas, retrasos en el proyecto.	Media	Alto	Alto
R05	Comunicación insuficiente con el tutor	Insatisfacción con el resultado final, menor calidad del proyecto, desviaciones de tiempo.	Baja	Alto	Medio
R06	Falta de capacitación en tecnologías específicas	Dificultades en la implementación de la base de datos por falta de	Baja	Alto	Medio

		conocimiento técnico.			

Una vez analizados los riesgos, proponemos una serie de medidas correctoras para evitar la aparición de una incidencia o, en caso de no ser posible, por lo menos minimizar su impacto.

Código	Acción	Riesgo Residual
A1R01	La planificación debe contar con un margen para imprevistos.	Medio
A2R01	Comunicar los problemas al tutor, solicitar ajustes de fechas.	Alto
A3R01	En caso de ser necesario, solicitar vacaciones en el trabajo para disponer de más tiempo para el desarrollo del proyecto.	Medio
A1R02	El análisis de requisitos debe hacerse en profundidad y documentarse de manera exhaustiva.	Medio
A1R03	Usar guardado en la nube para los documentos. Usar un sistema de control de versiones para los scripts. Almacenar frecuentemente copias de seguridad.	Bajo
A1R04	Establecer hitos intermedios para evaluar y reaccionar con rapidez a posibles desviaciones temporales.	Medio
A1R05	Establecer comunicaciones periódicas para indicar el progreso realizado y	Bajo

	comunicar los bloqueos producidos.	
A1R06	Buscar recursos en línea o pedir asesoramiento al tutor en caso de necesidad.	Bajo

1.4.3. Seguimiento del proyecto

Hito	Comentarios
PEC1	La entrega se ha realizado a tiempo y ha incluido todas las tareas previstas, por lo que no hay desviación de tiempos. Se proponen las siguientes correcciones que se incorporan para la PEC2: • Completar el resumen de la ficha del trabajo. • Añadir un vínculo a la referencia efectuada en la introducción. • Añadir un apartado de aspectos vinculados a la ética, sostenibilidad, responsabilidad social y/o derechos humanos y diversidad en la introducción. • Ajustar la presentación del Diagrama de Gantt para que sea visible desde el documento. • Añadir los hitos a los que corresponden las subtareas en el listado de tareas a realizar para mejorar su visibilidad.
PEC2	Si bien de manera interna hubo retrasos a la planificación prevista, se ha conseguido entregar en la fecha de entrega todas las tareas previstas sin desviación de tiempos en la planificación global del proyecto. Se proponen las siguientes correcciones que se incorporan para la PEC3: • Ampliar el contexto del problema a solucionar para que se pueda validar que los requisitos detectados surgen del mismo. • Incluir las palabras clave del proyecto. • Actualizar el índice. • Corrección de errata en el diagrama de Gantt: la fecha de fin de la PEC3 es el 20 de mayo y no el 21 de

	mayo. Se han observado también otros errores en fechas que se revisan para resolución. • Corrección de errata ortográfica en texto del requisito RF-17: eventos debe indicarse en plural. • Añadido de una columna en el listado de requisitos no funcionales que indique el tipo al que corresponde. • En el apartado de diseño conceptual del aplicativo (3.1.1), entidad “Event,” relación “Event has Location”, corrección de la errata al escribir la palabra recinto. • En el apartado de diseño conceptual del aplicativo (3.1.1), entidad “EventWithSales”, corregir la columna descripción del atributo “capacity” que contiene información incorrecta. • En el apartado de diseño conceptual del aplicativo (3.1.1), entidad Location, atributo id, corrección de la errata en la escritura de la palabra “recinto”. • Añadido de una explicación de la presentación de cada atributo en el apartado de diseño lógico (3.2) • Añadido de un apartado explicando los motivos de la elección del SGBD en el apartado de diseño físico (3.1). • Completado de términos y acrónimos más relevantes utilizados dentro de la memoria en el Glosario (apartado 5). Adicionalmente, realizamos las siguientes modificaciones: • Cambio del nombre de los campos “start” y “end” de Event al ser “end” un nombre reservado de PostgreSQL. • Cambios de nombres de identificadores y sus descripciones en el repositorio estadístico al no ser identificadores incrementales. • Cambio del nombre de la entidad “Top_Favorited_Events” por “Top_Favorite_Events” para adaptarlo a la gramática inglesa. • Modificación de la estructura de la tabla de Logs para asociar cada entrada a un procedimiento y creación de una tabla adicional que incluya el listado de procedimientos posible. • Modificación de la estructura de la tabla “Log” para permitir más valores (cambio a BIGSERIAL del identificador). • Modificación de la estructura de la tabla “Log” para añadir el usuario de la base de datos que efectúa la operación. • Añadir lógica de borrado en cascada para la entidad “Organizer_Contact” cuando se borra un organizador.

	• Añadir lógica en “Category” para que, cuando se borre una categoría padre, las categorías hijas automáticamente pasen a tener una referencia al padre nulas. • En “Category”, añadir una clave única que tenga en consideración la categoría padre y el nombre de la subcategoría, de manera que cada categoría tenga un nombre único cuando estén al mismo nivel. • En “Rating”, añadir borrado en cascada, de manera que cuando el usuario o el evento se borren, se pueda borrar la puntuación directamente. • En “Event_Favorite”, añadir borrado en cascada, de manera que cuando el usuario o el evento se borren, se pueda borrar el favorito directamente. • En “Event_Has_Category”, añadir borrado en cascada para cuando se elimine el evento. • En “Event_With_Sales”, añadir el borrado en cascada para cuando se elimine el evento. • En “Event_Change”, cambiamos el identificador por un BIGSERIAL para permitir una mayor cantidad de valores en la tabla. • En “Event”, añadir la columna “event_has_sales” para comprobar si la venta de entradas está o no activa. • En “Event_With_Sales”, añadimos una columna “event_sales” que contabiliza el número de ventas. Su valor por defecto es 0 y se actualizará cada vez que se añada una venta. • Modificar la referencia “event_id” en transaction en los diseños lógico y físico, que deberá corresponderse a “event_with_sales”.
PEC3	Durante la entrega de esta prueba, hemos sufrido ciertos retrasos y cambios de planificación en el desarrollo del producto. Esto es debido a que se priorizó la realización de pruebas unitarias para validar las funcionalidades desarrolladas en el producto y que ciertas fases de desarrollo han llevado más tiempo del inicialmente estipulado. Esto ha tenido como consecuencia la no inclusión del repositorio estadístico en esta entrega, tal y como estaba planeado. A priori, no se proponen correcciones sobre el desarrollo realizado. Proponemos y aplicamos los siguientes cambios respecto a la estructura propuesta inicialmente:

	• Añadimos un disparador para comprobar si se pueden añadir, o no, comentarios.
	• Añadimos una fecha a en la tabla “Transaction”, que por defecto será la fecha actual.
	• En EventStatistics:
	  ○ Cambiar el nombre de “Comments” por “RatingsCount”.
	  ○ Añadir una columna “TotalRating” que nos ayudará en el cálculo de la puntuación promedio ("AverageRating").
	  ○ Añadir una columna “Favorites” que nos ayudará a dictaminar cuáles son los eventos más marcados como favoritos.
	  ○ Borrar la columna “sales” al estar este dato duplicado en la tabla EventWithSales y poderse recuperar dichos datos directamente desde ahí.
	  ○ Cambiar el nombre de la columna “occupancy” por “occupation”
	• Eliminamos la entidad EventWithSalesStatistics al no ser necesaria para los cálculos de las estadísticas.
	• Añadimos las ventajas del uso de vistas en la sección de ventajas de PostgreSQL.
	• Modificamos los indicadores de enteros y porcentajes para adaptarlos a las necesidades de las estadísticas.
	• Añadimos la entidad “TopEventCities” que no estaba definida para el repositorio estadístico y necesitamos para poder obtener el listado de las 10 ciudades con mayor número de eventos.

1.5. Aspectos vinculados a la ética, sostenibilidad, responsabilidad social y/o derechos humanos

Por una parte, dado que recopilaremos datos de usuario, deberemos garantizar su correcta protección y almacenaje. Los datos de un usuario representan información propia y puede ser sensible, por ello, es importante ser estricto en el tratamiento, exposición y uso de esos datos y no recopilar más datos que los que sean estrictamente necesarios para el correcto funcionamiento del aplicativo.

Para la protección de estos datos de usuarios nos regiremos por el Reglamento General de Protección de Datos (RGPD) que es además un imperativo legal en la Unión Europea.

Por otra parte, aunque este trabajo se centra en el desarrollo e implementación en si de la solución técnica, si la administración de la base de datos fuera nuestra responsabilidad sería conveniente establecer un criterio de moderación que evitara la publicación de eventos que pudieran representar una estafa o un peligro para la comunidad o ciertos colectivos particularmente vulnerables.

Por último, es importante citar que, aunque es algo que comúnmente no se tiene en cuenta, los desarrollos que hacemos tienen un impacto ambiental: el tener servidores funcionando supone un gasto elevado de energía. Para minimizar el impacto que tiene nuestro proyecto a nivel de sostenibilidad.

Algunas de las medidas que podemos aplicar para mejorar la sostenibilidad en este proyecto son el uso de las tecnologías y algoritmos y métodos más recientes disponibles y el diseño de un sistema eficiente y optimizado para efectuar las consultas en el menor tiempo posible y de la mejor forma posible, usando la menor cantidad posible de recursos físicos y por tanto disminuyendo el consumo energético necesario para su ejecución.

1.6. Breve sumario de productos obtenidos

El objetivo principal de este proyecto es el desarrollo de un sistema con una base de datos relacional, que dispondrá de las siguientes características:

- Fichero README con las instrucciones para la ejecución del proyecto.
- Uso de una solución de orquestación de contenedores (Docker) que permita instalar el SGBD con todas sus dependencias con facilidad e independientemente de las características del entorno de ejecución del proyecto.
- Scripts de inicialización de la base de datos. Ejecución de estos scripts de forma automática desde Docker.
- Uso del framework para pruebas unitarias de PostgreSQL pgTAP. Scripts para pruebas de las funcionalidades desarrolladas.
- Uso de la extensión pg_cron para programar tareas.

1.7. Breve descripción de los otros capítulos de la memoria

Durante los siguientes capítulos de este Trabajo Final de Grado analizaremos los siguientes contenidos:

- **Requisitos del producto**: partiendo del contexto de realización de este trabajo y la idea de desarrollo propuesta, analizaremos cuáles son las características que debe cumplir nuestro aplicativo, tanto en vistas de funcionalidad como de calidad. Por otra parte, elegiremos qué sistema de gestión de bases de datos (SGBD) usaremos en el desarrollo del proyecto

- **Diseño**: una vez sepamos cuáles son los requisitos que necesita nuestro aplicativo, tendremos que diseñar cómo estructuramos y organizamos nuestra base de datos. Lo haremos en tres diferentes niveles:
    - **Conceptual**: representando diferentes entidades y las relaciones entre ellas.

    - **Lógico**: se transforma el esquema conceptual en un esquema lógico, definiendo la relación entre los diferentes datos de manera independiente del sistema de gestión de base de datos elegido.
    - **Físico**: definimos cómo se deben almacenar los datos, teniendo además en cuenta las características del SGBD elegido.

- **Implementación**: una vez tengamos el diseño finalizado empezaremos a desarrollar el producto. Usando una solución de orquestación de contenedores (Docker), crearemos un entorno en el que se instalará PostgreSQL. Inicializaremos la base de datos, creando diferentes espacios y usuarios, crearemos las diferentes tablas que contendrán toda la información y definiremos procesadores y disparadores que ayuden a implementar la lógica de negocio. Por último, incorporaremos una serie de pruebas automatizadas que nos permitirán probar y validar los requisitos que hemos definido para nuestra base de datos.

2. Requisitos del producto

Tal y como hemos tratado anteriormente, y por la metodología elegida, la correcta obtención y gestión de los requisitos es crucial para el correcto desarrollo del proyecto, ya que el modelo en cascada es poco susceptible a cambios. Por tanto, este apartado es crucial para el desarrollo del proyecto.

Podemos definir los requisitos del producto como las necesidades o restricciones que los diferentes *stakeholders* tienen sobre el producto. Un *stakeholder* es toda aquella persona o entidad que tiene un impacto o interés sobre el sistema [6].

En este caso, el trabajo es desarrollado de forma individual y sin la participación de empresas o interesados externos más allá de la institución académica. Por tanto, asumimos algunos de los papeles de dichos *stakeholders* asumiendo sus puntos de vista para la toma de requisitos.

Algunos de los principales *stakeholders* del proyecto son:

- **El tutor**: se encarga de la supervisión y seguimiento del proyecto, además, indica cuáles son los requisitos mínimos del proyecto para que este pueda ser completado y entregado exitosamente.

- **Tribunal académico**: el tribunal académico evaluará el proyecto presentado y los conocimientos aplicados en éste.

- **Usuarios finales**: usarán el sistema que explote la base de datos implementada. Aunque no están presentes a lo largo del desarrollo del proyecto, asumimos su papel.

- **Gestores de eventos**: están interesados en que sus eventos se muestren y promocionen en la plataforma derivada de este proyecto. Al igual que los usuarios finales, no están presentes a lo largo del desarrollo del proyecto y asumimos su papel.

Entender los puntos de vista de los *stakeholders* nos permitirá obtener y gestionar los diferentes requisitos que deberá tener nuestro sistema.

2.1. Requisitos funcionales

Los requisitos funcionales, como lo indica su nombre, son aquellos que hacen referencia a la funcionalidad y a los datos que debe cumplir nuestro proyecto.

Código	Requisito
RF01	Debe almacenarse un listado de recintos, con su nombre y ubicación, de manera que puedan ser geolocalizados.

RF02	Debe almacenarse un listado de categorías y subcategorías de evento.
RF03	Debe poder almacenarse un listado de organizadores y promotores de eventos y sus datos de contactos. Opcionalmente, en caso de tener contacto directo con la empresa, pueden almacenarse los datos de las personas de contacto.
RF04	Debe almacenarse un listado de eventos con todos sus datos y detalles: nombre, ubicación (recinto), empresa organizadora, fechas de inicio y fin, horarios, descripción, categorías, precio, si acepta ventas de entradas desde la plataforma y, opcionalmente, una imagen.
RF05	Dado que la base de datos servirá para una plataforma de publicación de eventos, debemos poder distinguir entre los estados de publicación del evento: • **Publicado**: es visible para el público y a través de la plataforma. • **Borrador**: se están introduciendo los datos del evento y por tanto aún no está disponible a través de la plataforma.
RF06	Debemos poder distinguir el estado de disponibilidad del evento: • **Programado**: el evento está programado para las fechas dadas. • **Cancelado**: el evento no va a tener lugar tal y como estaba planteado.
RF07	Debemos guardar un registro de los cambios que se producen en la planificación de cada evento. Los tipos de cambio que se pueden producir son los siguientes: • **Aplazado** • **Cancelado** • **Cambio de recinto** • **Cambio de precio** • **Otros** Para cada cambio producido en la planificación de un evento debemos almacenar cuál es el cambio que se ha producido y en qué fecha ha tenido lugar.
RF08	Cuando un evento cambia de estado a cancelado, automáticamente debe paralizarse la venta de entradas.
RF09	Cuando el sistema de venta está activado para un evento, debe establecerse un número total de entradas que se pueden vender

	por evento.
RF10	Debe poder almacenarse un listado con los usuarios que se registran, incluyendo datos básicos de contacto (nombre, apellidos y correo electrónico) y una contraseña para que puedan iniciar sesión.
RF11	Debemos poder distinguir los usuarios del sistema en función de los roles que desempeñan en el sistema: **administradores** o **clientes**.
RF12	Los usuarios deben poder guardar eventos como favoritos.
RF13	Para cada venta efectuada, debe poder guardarse un registro de las transacciones efectuadas por cada usuario, que incluirá el evento al que corresponden, el precio unitario por entrada, el número de entradas compradas y una referencia de la compra efectuada
RF14	La referencia de compra debe ser un código alfanumérico de 5 caracteres y debe ser único para cada usuario.
RF15	Cuando un evento ha vendido todas las entradas permitidas por el aforo debe paralizarse la venta de entradas.
RF16	Debe permitirse y optimizarse la realización de consultas avanzadas de búsqueda de eventos, de manera que los eventos puedan filtrarse por categorías, rangos de precio, ubicación y fecha.
RF17	Los usuarios deben poder añadir valoraciones numéricas (sobre 5) y comentarios a los eventos.
RF18	Los comentarios pueden desactivarse en función del evento.
RF19	Como funcionalidad de moderación, los comentarios tendrán que ser aprobados previamente a ser publicados.
RF20	Un usuario únicamente puede publicar un comentario y valoración por evento.
RF21	Debe crearse un repositorio estadístico con los siguientes indicadores, que deberán calcularse y almacenarse a través de procedimientos automatizados: • Listado de los 10 eventos con más comentarios. • Listado de los 10 eventos con mejor valoración. • Listado de los 10 eventos con más ventas. • Listado de los 20 recintos con más eventos. • Listado de los 20 eventos más marcados como favoritos por los usuarios.

- Listado de las 10 ciudades con mayor número de eventos.
- Precio medio de los eventos de pago.
- Número total de usuarios no administradores registrados en el sistema.
- Promedio del número de transacciones por usuario del sistema.
- Porcentaje de variación del número de transacciones durante el último mes respecto al anterior.
- Porcentaje medio de ocupación de los eventos con venta de entradas.
- Porcentaje de eventos con entradas agotadas.
- Porcentaje de eventos con el sistema de pago activo.

Debemos tener en cuenta que otros requisitos, como pueden ser la encriptación y seguridad de las contraseñas almacenadas, no se plantean en este proyecto ya que se consideran que son responsabilidad del aplicativo que explota la base de datos, cuyo diseño y desarrollo no está planteado en este trabajo, y no de la base de datos en sí.

2.2. Requisitos no funcionales

Por otra parte, los requisitos no funcionales son aquellos que hacen referencia a calidades esperadas del sistema.

Código	Requisito	Tipo
RNF01	Se implementará una base de datos relacional.	Cumplimiento
RNF02	Se implementará una base de datos transaccional.	Cumplimiento
RNF03	Se usará PostgreSQL como motor de base de datos.	Cumplimiento
RNF04	Se incorporará un diseño escalable para facilitar futuras modificaciones.	Mantenimiento y soporte
RNF05	La base de datos debe poder gestionar cualquier volumen de datos.	Mantenimiento y soporte
RNF06	Se usarán procedimientos de alta, baja y modificación (ABM) en todas las entidades relevantes.	Cumplimiento
RNF07	Se usará un diagrama UML para la realización del diseño conceptual de la base de datos, con todas las entidades y sus relaciones.	Cumplimiento

RNF08	Debe almacenarse en un registro todas las llamadas a procedimientos que se realicen, incluyendo el nombre del procedimiento y sus parámetros de entrada y de salida.	Cumplimiento
RNF09	Los procedimientos dispondrán de un parámetro de salida llamado RSP, de tipo “string”, que indicará si la ejecución ha finalizado correctamente (valor “OK”) o si ha fracasado (valor “ERROR+TIPO DE ERROR”)	Cumplimiento
RNF10	Los procedimientos dispondrán de tratamiento de errores.	Mantenimiento y soporte
RNF11	Se desarrollarán scripts que permitan probar la funcionalidad y operabilidad de la base de datos.	Mantenimiento y soporte
RNF12	El repositorio estadístico debe ofrecer los resultados en tiempo constante 1.	Cumplimiento
RNF13	No deben usarse funciones de agregación con group by para el almacenado de datos en el repositorio estadístico.	Cumplimiento

2.3. Elección del SGBD: PostgreSQL

El Sistema Gestor de Bases de Datos (SGBD) es un software diseñado para administrar una base de datos, proporcionando diferentes herramientas que nos permitirán efectuar todo tipo de operaciones en ella.

Teniendo en cuenta las características del proyecto y mi experiencia personal como desarrollador, hemos optado por el uso de una base de datos relacional (que es aquella en la que los datos están relacionados entre sí).

Entre las diferentes opciones de SGBD, hemos decidido desarrollar el proyecto en PostgreSQL por los siguientes motivos:

- **Licencia de código abierto**: PostgreSQL es gratuito y de código abierto y se permite su uso y modificación sin restricciones.

- **Comunidad**: Además de ser de código abierto, PostgreSQL es uno de los SGBD más usados y por tanto dispone de una amplia comunidad de desarrolladores. El hecho de tener una amplia comunidad facilita el acceso a documentación y soporte en caso de ser necesario.

- **Base de datos objeto-relacional**: PostgreSQL es una base de datos de tipo objeto-relacional y no puramente relacional, lo que permite usar una amplia cantidad de tipos de datos que nos pueden ser de ayuda en la

implementación de la funcionalidad requerida.

- **Soporte multiplataforma**: PostgreSQL puede ejecutarse en prácticamente todas las versiones de sistemas operativos como Linux, Unix o Windows, lo que da una gran versatilidad a la hora de desplegar y desarrollar.

- **Extensibilidad**: el desarrollador puede definir sus propios tipos de datos, operadores y funciones, además de poder incorporar funcionalidades adicionales a través de extensiones. Esto nos da una gran flexibilidad a la hora de gestionar la base de datos.

- **Soporte ACID**: PostgreSQL soporta transacciones ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), lo que nos garantiza la integridad de los datos en todas las operaciones y nos proporciona una gran estabilidad y fiabilidad.

- **Procedimientos almacenados y disparadores**: PosgreSQL soporta el uso de procedimientos almacenados y disparadores, cuyo uso es esencial para cumplir varios de los requisitos especificados. Además, da soporte a lenguajes procedurales, como puede ser el caso de PL/pgSQL. En el caso de MySQL, por ejemplo, el soporte es mucho más limitado.

- **Rendimiento**: PostgreSQL puede usarse para manejar un volumen amplio de datos y soporta la indexación de datos, lo cual puede mejorar el rendimiento de las consultas que se efectúan. Además, es idóneo para ser usado en sistemas a gran escala.

- **Soporte con Docker**: Existe una imagen oficial de Docker que nos permitirá usar contenedores, lo cual nos da mucha versatilidad a la hora de configurar y trabajar con PostgreSQL en nuestro entorno local y de poder ejecutar el proyecto en otras máquinas.

- **Uso de vistas**: El uso de vistas nos permitirá simplificar algunas de las consultas efectuadas al repositorio estadístico.

- **Interés personal**: En mi caso personal, he trabajado sobre todo con MySQL, que es uno de los SGBD más usados a nivel mundial, pero algunas de las anteriormente citadas características de PostgreSQL hacen que me parezca más interesante y pueda usar este trabajo como una oportunidad de aprender a manejarme con un nuevo SGBD.

Las ventajas que hemos citado son únicamente algunas de las que hemos considerado más relevantes teniendo en cuenta las características de este proyecto, si bien no son las únicas de las que dispone PostgreSQL.

En resumen, PostgreSQL es una solución que, no únicamente satisface mi interés personal por trabajar más con ella para estudiar su comportamiento, sino que nos permite cumplir todos los requisitos de nuestro proyecto.

3. Diseño

3.1. Diseño conceptual

El primer paso de diseño es la realización de un diseño conceptual que nos permitirá visualizar las diferentes entidades que forman parte de nuestro modelo y las relaciones entre ellas.

Para conseguir nuestro objetivo, utilizaremos esquemas UML (Unified Modeling Language).

3.1.1. Diseño del aplicativo

En este primer diseño conceptual tenemos en cuenta aquellos aspectos relacionados con el aplicativo en el que gestionaremos aspectos relacionados con la gestión de los eventos, el sistema de usuarios y la venta de entradas:

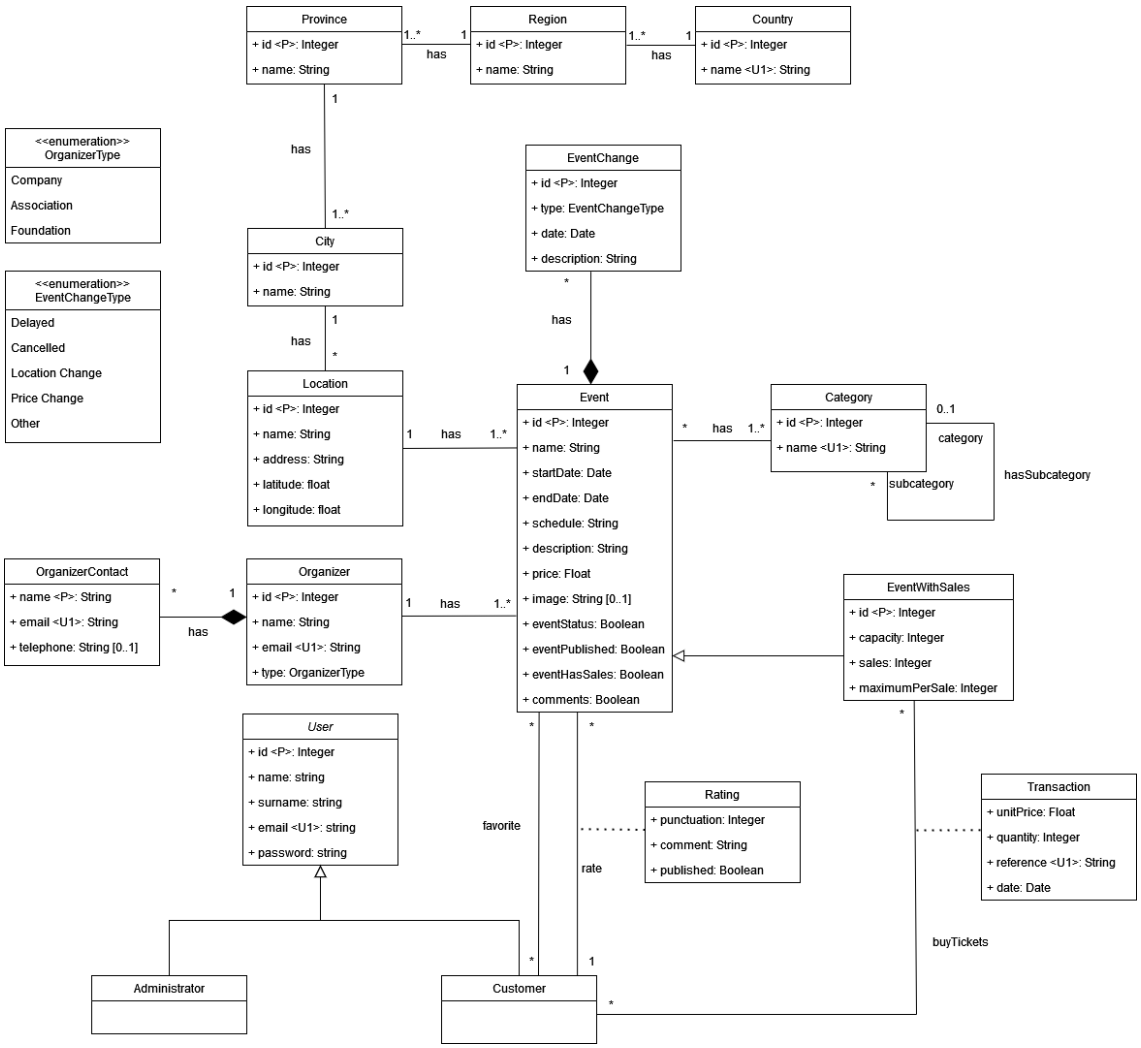


Figura 2: Diseño conceptual del aplicativo

Event: esta entidad contiene un listado de los eventos almacenados en el sistema.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar a los eventos de manera única.
name	String	Nombre del evento.
startDate	Date	Fecha de inicio del evento.
endDate	Date	Fecha de fin del evento.
schedule	String	Horarios en los que tendrá lugar el evento.
description	String	Descripción del evento.
price	Float	Precio del evento.
image	String	Atributo opcional que permite asociar una imagen al evento.
eventPublished	Boolean	Indica si el evento está publicado (valor verdadero) o se encuentra en estado de borrador (falso).
eventHasSales	Boolean	Indica si el evento tiene activa la venta de entradas (valor verdadero) o no (valor falso).
eventStatus	Boolean	Indica si el evento sigue programado (verdadero) o si ha sido cancelado (falso).
comments	Boolean	Indica si los comentarios y valoraciones están permitidos o no para el evento.

Relaciones:

Relación	Cardinalidad	Descripción
Event has Category	*:1..*	Un evento puede tener asociadas una o varias categorías y subcategorías.
Event has Organizer	1..*:1	Un evento debe tener asociado un único organizador.
Event has Location	1..*:1	Un evento debe tener asociado un único recinto.

Event has EventChange	1:*	Un evento puede tener un número ilimitado de cambios asociados.
Event is Customer favorite	*:*	El evento puede ser marcado como favorito por un número ilimitado de clientes.
Event is rated by Customer	*:1	El evento puede ser valorado una única vez por un usuario en caso de que los comentarios estén permitidos. Esta relación tiene una entidad asociativa Rating en el que se almacenan las valoraciones y comentarios dadas por los usuarios.

EventWithSales: es una entidad hija de Event y corresponde a los eventos que tienen el sistema de ventas de la plataforma activo.

Atributo	Tipo	Descripción
Id	Integer	Clave primaria de la tabla y permite identificar a los eventos con venta de manera única.
capacity	Integer	Aforo total del evento, indica el número total de billetes que se pueden vender.
sales	Integer	Número total de entradas vendidas.
maximumPerSale	Integer	Indica el número máximo de billetes que se pueden comprar por transacción.

Relaciones:

Relación	Cardinalidad	Descripción
EventWithSell is bough by Customer	*:*	Un evento con las ventas activas puede tener ninguna, una o más ventas asociadas.

EventChange: esta entidad contiene un listado de los cambios producidos en un evento.

Atributo	Tipo	Descripción
id	Integer	Permite identificar los cambios en un evento.

type	EventChangeType (Enum)	Indica el tipo de cambio que se produce en el evento.
date	Date	Fecha del cambio en el evento.
description	String	Descripción del cambio producido.

Relaciones:

Relación	Cardinalidad	Descripción
EventChange has Event	*:1	Cada cambio de evento debe estar asociado a un evento. Cuando se elimina el evento, deben eliminarse también todos sus cambios asociados.

Category: esta entidad contiene las diferentes categorías y subcategorías en las que se pueden clasificar los eventos.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar a las categorías de manera única.
name	String	Nombre de la categoría.

Relaciones:

Relación	Cardinalidad	Descripción
Category has Event	1..*:*	Cada categoría debe tener al menos un evento asociado.
Category has Subcategory	0..1:*	Las subcategorías son un tipo específico de categoría que tienen una categoría padre. Una categoría puede ser subcategoría de otra o no. Se ha definido esta relación como una asociación recursiva.
Subcategory has Category	*:0..1	Cada subcategoría debe esta asociada a una única categoría padre. Sin embargo, las categorías padres no tienen por qué tener ninguna categoría padre.

Organizer: esta entidad contiene los diferentes organizadores de eventos presentes en el sistema.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar al organizador de manera única.
name	String	Nombre del organizador del evento.
email	String	Correo electrónico de contacto del organizador del evento.
type	OrganizerType (Enum)	Indica el tipo de organizador del evento (si es una empresa o una asociación

Relaciones:

Relación	Cardinalidad	Descripción
Organizer has Event	1:1..*	Cada organizador debe tener al menos un evento asociado.
Organizer has OrganizerContact	1:*	Cada organizador puede tener, opcionalmente, personas de contacto asociadas.

OrganizerContact: esta entidad contiene los datos de una persona de contacto de un organizador de eventos.

Atributo	Tipo	Descripción
name	String	Nombre de la persona de contacto del organizador.
email	String	Correo electrónico de contacto.
Telephone	String	Número de teléfono opcional.

Relaciones:

Relación	Cardinalidad	Descripción
OrganizerContact has Organizer	*:1	Cada contacto debe estar asociado únicamente a un organizador. En caso de que el organizador se elimine, la existencia del contacto no tiene sentido.

Location: esta entidad contiene los datos de los recintos en los que se organizan eventos.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar al recinto de manera única.
name	String	Nombre del recinto.
Address	String	Dirección del recinto.
Latitude	Float	Coordenadas del recinto: este campo indica la latitud.
Longitude	Float	Coordenadas del recinto: este campo indica la longitud.

Relaciones:

Relación	Cardinalidad	Descripción
Location has Event	1:1..*	Cada evento debe tener por lo menos un recinto asociado.
Location has City	*:1	Cada recinto debe tener una ciudad asociada.

City: esta entidad contiene los datos de las ciudades en las que se albergan eventos.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar la ciudad de manera única.
name	String	Nombre de la ciudad.

Relaciones:

Relación	Cardinalidad	Descripción
City has location	1:*	Las ciudades pueden estar asociadas a un recinto.
City has province	1..*:1	Cada ciudad debe estar asociada a una provincia.

Province: esta entidad contiene los datos de las provincias en las que se albergan eventos.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar la provincia de manera única.
name	String	Nombre de la provincia.

Relaciones:

Relación	Cardinalidad	Descripción
Province has city	1:1..*	Las provincias pueden tener asociadas una o más ciudades.
Province has Region	1..*:1	Cada ciudad debe estar asociada a una región.

Region: esta entidad contiene los datos de las regiones en las que se albergan eventos.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar la región de manera única.
name	String	Nombre de la región.

Relaciones:

Relación	Cardinalidad	Descripción
Region has Province	1:1..*	Las regiones pueden tener asociadas una o más provincias.
Province has Country	1..*:1	Cada región debe estar asociada a un país.

Country: esta entidad contiene los datos de los países en los que se albergan eventos.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar al

		país de manera única.
name	String	Nombre del país.

Relaciones:

Relación	Cardinalidad	Descripción
Country has Region	1:1..*	Los países pueden tener asociadas una o más regiones.

User: se trata de una clase abstracta que contiene los datos de los diferentes usuarios del sistema. Estos usuarios podrán ser de tipo Administrador o Cliente.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar al usuario de manera única.
name	String	Nombre del usuario.
surname	String	Apellido del usuario.
email	String	Correo electrónico del usuario.
password	String	Contraseña del usuario.

Administrator: es un usuario con permisos para administrar el sistema. Hereda de la clase User.

Customer: el cliente es el tipo de usuario con derecho a consumir los datos públicos del sistema y a interactuar con él para diferentes acciones, como puede ser la compra de entradas. Hereda de la clase User.

Relaciones:

Relación	Cardinalidad	Descripción
Customer has favorite Event	*:*	Los usuarios pueden marcar como favoritos la cantidad de eventos que consideren.
Customer rates Event	1:*	Los usuarios pueden valorar eventos. Esta relación tiene una entidad asociativa Rating que albergará la puntuación y comentario dados por el usuario.

Customer buys tickets of EventWithSales	*:*	Un cliente puede comprar un número indeterminado de entradas a un evento con el sistema de ventas activo.

Rating: se almacena una puntuación dada por un usuario a un evento.

Atributo	Tipo	Descripción
punctuation	Integer	Puntuación, sobre 5, dada por el usuario.
comment	String	Comentario del usuario al evento.
published	Boolean	Indica si el comentario ha sido aprobado y, por tanto, está disponible públicamente.

Relaciones:

Relación	Cardinalidad	Descripción
Asociativa en la relación rate		Está siempre asociado a una relación entre la entidad Customer y la entidad Event.

Transaction: se almacena una transacción de compra de entradas de un evento con pago realizada por un usuario.

Atributo	Tipo	Descripción
unitPrice	Float	Precio unitario de las entradas compradas.
quantity	Integer	Cantidad de entradas compradas.
reference	String	Referencia única de la compra.
date	Date	Fecha de realización de la transacción.

Relaciones:

Relación	Cardinalidad	Descripción
Asociativa en la relación buyTickets		Está siempre asociado a una relación entre la entidad Customer y la entidad EventWithSales.

En la realización de este diseño, se ha decidido no usar composición entre las entidades Country, Region, Province y City ya que representan un espacio físico que no debe ser borrado. Sin embargo, en las propias relaciones entre las clases se ha especificado la necesidad de que una ciudad tenga una provincia asociada, una provincia tenga una región asociada y una región tenga un país asociado.

Por otra parte, hemos planteado los identificadores como enteros que se autoincrementan al insertar registros en la base de datos.

3.1.2. Diseño del repositorio estadístico

Tal y como se han indicado en los requisitos, el repositorio estadístico deberá tener un tiempo constante de lectura 1.

Por ello, crearemos una única entidad en la que almacenaremos los resultados que debemos devolver en las consultas.

Debido a la restricción que disponemos que nos impide usar funciones de agregación (RNF-13), usaremos una serie de tablas auxiliares que nos permitirán almacenar la información que necesitaremos para efectuar los cálculos correspondientes:

- Una tabla de eventos, con los siguientes datos:
  - Número de comentarios.
  - Valoración media.
  - Valoración total.
  - Número de veces marcado como favorito.
  - Ocupación del evento.
- Una tabla de recintos, con los siguientes datos:
  - Número de eventos.
- Una tabla de ciudades, con los siguientes datos:
  - Número de eventos.
- Una tabla de transacciones, con los siguientes datos:
  - Año.
  - Mes.
  - Número de transacciones.
- Una tabla auxiliar con contadores generales del sistema, con los siguientes datos:
  - Número total de usuarios no administradores en el sistema.
  - Número total de eventos de pago en el sistema.
  - Número total de eventos en el sistema.
  - Número total de transacciones del sistema.

Conociendo estos datos, podremos desarrollar el diseño conceptual del repositorio estadístico:

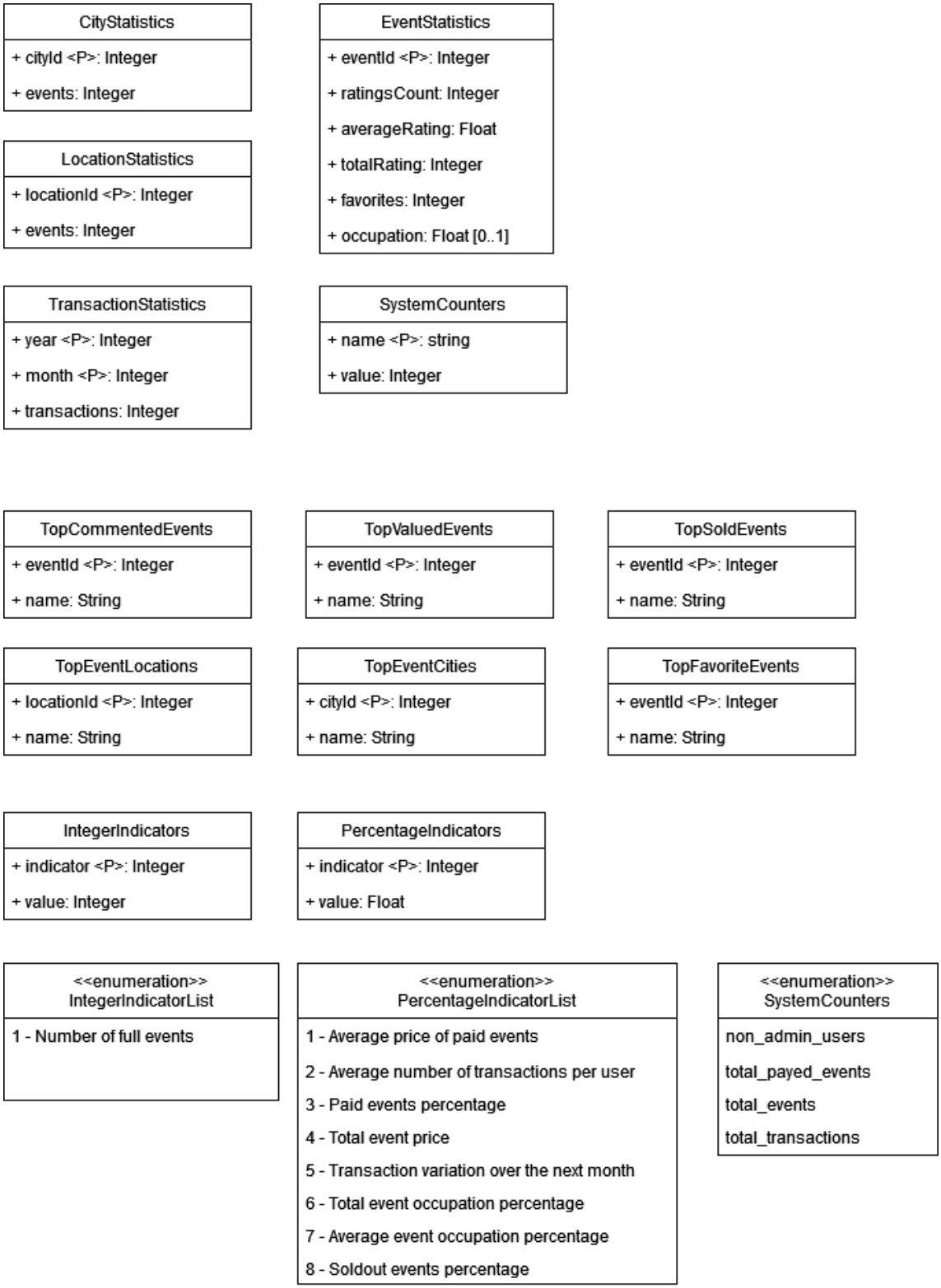


Figura 3: Diagrama conceptual del repositorio estadístico

EventStatistics: esta entidad contiene estadísticas de cada uno de los eventos de la base de datos.

Atributo	Tipo	Descripción
eventId	Integer	Clave primaria de la tabla, se corresponde con cada uno de los eventos de la base de datos.
ratingsCount	Integer	Número total de comentarios en el evento.
averageRating	Float	Promedio de las valoraciones en el evento.
totalRating	Integer	Total de la suma de valoraciones en el evento.
favorites	Integer	Número de veces marcado favorito por los usuarios.
occupation	Float|Null	Porcentaje de ocupación del evento.

CityStatistics: esta entidad contiene estadísticas de cada una de las ciudades de la base de datos.

Atributo	Tipo	Descripción
cityId	Integer	Clave primaria de la tabla, se corresponde con cada una de las ciudades de la base de datos.
events	Integer	Número total de eventos en la ciudad.

LocationStatistics: esta entidad contiene estadísticas de cada uno de los recintos de la base de datos.

Atributo	Tipo	Descripción
locationId	Integer	Clave primaria de la tabla, se corresponde con cada uno de los recintos de la base de datos.
events	Integer	Número total de eventos en el recinto.

TransactionStatistics: esta entidad contiene estadísticas de las transacciones efectuadas.

Atributo	Tipo	Descripción
year	Integer	Año.

month	Integer	Mes.
transactions	Integer	Número de transacciones efectuadas en el año y mes dados.

SystemCounters: esta entidad contiene diferentes contadores generales del sistema que pueden ser de interés.

Atributo	Tipo	Descripción
name	String	Nombre del contador. Deben corresponderse con alguno de los valores de SystemCounters.
value	Integer	Valor del contador seleccionado.

Y, por otra parte, las entidades con los resultados:

TopCommentedEvents: disponemos un listado con los eventos con mayor número de comentarios. Obtenemos estos datos de EventStatistics.

Atributo	Tipo	Descripción
eventId	Integer	ID del evento.
name	String	Nombre del evento.

TopValuedEvents: disponemos un listado con los eventos con mayor valoración. Obtenemos estos datos de EventStatistics.

Atributo	Tipo	Descripción
eventId	Integer	ID del evento.
name	String	Nombre del evento.

TopSoldEvents: disponemos un listado con los eventos con mayor número de ventas. Obtenemos estos datos de EventWithSaleStatistics.

Atributo	Tipo	Descripción
eventId	Integer	ID del evento.
name	String	Nombre del evento.

TopFavoriteEvents: disponemos un listado con los eventos que más usuarios han marcado como favoritos. Obtenemos estos datos de EventStatistics.

Atributo	Tipo	Descripción
eventId	Integer	ID del evento.
name	String	Nombre del evento.

TopEventLocations: disponemos un listado con los recintos con mayor número de eventos organizados en ellos. Obtenemos estos datos de LocationStatistics.

Atributo	Tipo	Descripción
locationId	Integer	ID del recinto.
name	String	Nombre del recinto.

TopEventCities: disponemos un listado con las ciudades con mayor número de eventos organizadas en ellos. Obtenemos estos datos de CityStatistics.

Atributo	Tipo	Descripción
cityId	Integer	ID de la ciudad.
name	String	Nombre de la ciudad.

El resto de los indicadores son valores enteros o porcentuales, por lo que hacemos dos tablas en función del tipo de datos que almacenen:

IntegerIndicators: son un listado de los diferentes indicadores con un valor entero.

Atributo	Tipo	Descripción
indicator	Integer	ID del indicador representado. Viene representado en el enumerador IntegerIndicatorList.
value	Integer	Valor del indicador representado.

PercentageIndicators: son un listado de los diferentes indicadores con un valor decimal.

Atributo	Tipo	Descripción
indicator	Integer	ID del indicador representado. Viene representado en el enumerador PercentageIndicatorList.
value	Float	Valor del indicador representado.

3.1.3. Diseño del registro

Tal y como se especifica en los requisitos del proyecto, deberemos almacenar un registro de las acciones más importantes realizadas sobre la base de datos.

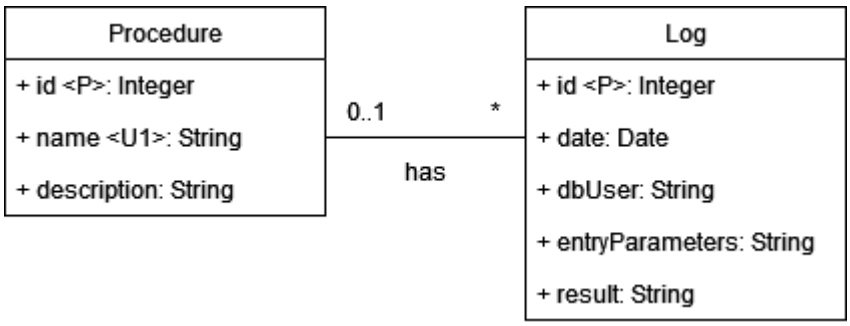


Figura 4: Diagrama conceptual del diseño del registro

Log: almacena un listado de los procesos ejecutados, con sus parámetros de entrada y salida y la fecha de ejecución.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar al registro de manera única.
date	Date	Fecha del evento.
entryParameters	String	Parámetros de entrada del proceso que

		referencia el registro
result	String	Resultado de la ejecución del proceso.

Relaciones:

Relación	Cardinalidad	Descripción
Log has Procedure	0..1:1	Los registros pueden estar asociados o no a un procedimiento. El caso de un registro no asociado a un procedimiento se da, fundamentalmente, cuando al ejecutarse el proceso este no está dado de alta en la tabla de procesos.

Procedure: almacena un listado de los procedimientos dados de alta en la base de datos.

Atributo	Tipo	Descripción
id	Integer	Clave primaria de la tabla y permite identificar al proceso de manera única.
name	String	Nombre del proceso.
description	String	Descripción del proceso.

Relaciones:

Relación	Cardinalidad	Descripción
Procedure has Log	0..1:*	Cada proceso puede tener un número indeterminado de eventos de registro asociados.

3.2. Diseño lógico

Una vez completado el diseño conceptual, usamos el modelo UML generado y lo transformaremos en un modelo lógico relacional.

En la presentación del diseño lógico, usamos los siguientes códigos:

- Los atributos subrayados se corresponden con la clave primaria.
- Los atributos marcados en negrita se corresponden con atributos no nulos.

- Los atributos subrayados con línea discontinua se corresponden con atributos únicos (y constituyen, por tanto, una clave alternativa).

Para el aplicativo:

Country (id, **name**)

Region (id, **name**, **country_id**)
{country_id} is foreign key to Country

Province (id, **name**, **region_id**)
{region_id} is foreign key to Region

City (id, **name**, **province_id**)
{province_id} is foreign key to Province

Location (id, **name**, **city_id**, **address**, **latitude**, **longitude**)
{city_id} is foreign key to City

Category (id, **name**, parent_category)
{parent_category} is foreign key to Category

Organizer (id, **name**, **email**, **type**)

OrganizerContact (name, organizer_id, **email**, telephone)
{organizer_id} is foreign key to Organizer

Event (id, **name**, **start_date**, **end_date**, **schedule**, **description**, **price**, image, **event_status**, **event_published**, **event_has_sales**, **comments**, **organizer_id**, **location_id**)
{organizer_id} is foreign key to Organizer
{location_id} is foreign key to Location

EventWithSales (id, **event_id**, **capacity**, **sales**, **maximum_per_sale**)
{event_id} is foreign key to Event

EventHasCategory (event_id, category_id)
{event_id} is foreign key to Event
{category_id} is foreign key to Category

EventChange (id, event_id, **type**, **date**, **description**)
{event_id} is foreign key to Event

User (id, **name**, **surname**, **email**, **password, roles**)

EventFavorite (event_id, customer_id)
{event_id} is foreign key to Event
{customer_id} is foreign key to User

Rating (user_id, event_id, **punctuation**, **comment**, **published**)
{user_id} is foreign key to User
{event_id} is foreign key to Event

Transaction (user_id, event_id, **unit_price**, **quantity**, **reference**, **date**)
{user_id} is foreign key to User
{event_id} is foreign key to Event

Para el repositorio estadístico:

CityStatistics (city_id, **events**)

LocationStatistics(location_id, **events**)

EventStatistics (event_id, **ratings_count**, **average_rating**, **total_rating**, **favorites**, occupation)

TransactionStatistics (year, month, **transactions**)

SystemCounters (name, **value**)

TopCommentedEvents (event_id, **name**)

TopValuedEvents (event_id, **name**)

TopSoldEvents (event_id, **name**)

TopEventLocations (location_id, **name**)

TopEventCities (city_id, **name**)

TopFavoriteEvents (event_id, **name**)

IntegerIndicators (indicator, **value**)

PercentageIndicators (indicator, **value**)

Para los registros:

Log (id, **date**, procedure_id, **entry_parameters**, **result**)
{procedure_id} is foreign key to Procedure

Procedure (id, **name**, **description**)

En esta fase hemos tomado las siguientes consideraciones de diseño:

- En el caso de los usuarios, aunque tenemos usuarios de tipo administrador y cliente, distinguimos el tipo a través de un campo “roles”, ya que ninguna de las dos entidades tiene datos adicionales que necesitemos cargar.

- Hemos decidido que, en el caso de EventWithSales, la tabla tenga un ID propio que permita una mayor flexibilidad del modelo usado.

3.3. Diseño físico

Una vez completados los diseños conceptuales y lógicos y teniendo en cuenta la elección de PostgreSQL como SGBD, completamos el diseño físico de la siguiente forma:

Para el aplicativo:

Country				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SMALLSERIAL	No	Identificador único autoincremental para el país.
	name	VARCHAR(100)	No	Nombre del país.
UNIQUE (name)				

Region				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SERIAL	No	Identificador único autoincremental para la región.
	name	VARCHAR(100)	No	Nombre de la regióm.
	country_id	SMALLINT	No	Identificador del país al que pertenece la región.
FOREIGN KEY (country_id) REFERENCES Country (id)				

Province				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SERIAL	No	Identificador único autoincremental para la provincia.
	name	VARCHAR(100)	No	Nombre de la provincia.
	region_id	INTEGER	No	Identificador de la región a la que pertenece la provincia.
FOREIGN KEY (region_id) REFERENCES Region (id)				

City				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SERIAL	No	Identificador único autoincremental para la ciudad.
	name	VARCHAR(100)	No	Nombre de la ciudad.
	province_id	INTEGER	No	Identificador de la provincia a la que pertenece la ciudad.
FOREIGN KEY (province_id) REFERENCES Province (id)				

Location				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SERIAL	No	Identificador único autoincremental para el recinto.
	name	VARCHAR(255)	No	Nombre del recinto.
	city_id	INTEGER	No	Identificador de la ciudad en la que se encuentra el recinto.
	address	VARCHAR(255)	No	Dirección del recinto.
	latitude	REAL	No	Latitud de la ubicación del recinto.
	longitude	REAL	No	Longitud de la ubicación del recinto.
FOREIGN KEY (city_id) REFERENCES City (id)				
INDEX idx_geolocation ON Location (latitude, longitude)				

Category				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SERIAL	No	Identificador único autoincremental para la categoría.
	name	VARCHAR(255)	No	Nombre de la categoría.
	parent_category	INTEGER	Sí	Categoría padre, si existe.
FOREIGN KEY (parent_category) REFERENCES Category (id) ON DELETE SET NULL				

CREATE UNIQUE INDEX idx_unique_category ON category(name, parent_category) WHERE parent_category IS NOT NULL
CREATE UNIQUE INDEX idx_unique_category_null_parent ON caregory(name) WHERE parent_category IS NULL

Organizer				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SERIAL	No	Identificador único autoincremental para el organizador.
	name	VARCHAR(255)	No	Nombre del organizador.
	email	VARCHAR(255)	No	Correo electrónico del organizador.
	type	VARCHAR(20)	No	Tipo de organizador
UNIQUE (email)				
CHECK type IN (‘Company’, ‘Association’, ‘Foundation’)				

Organizer_Contact				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	name	VARCHAR(255)	No	Nombre del contacto de la organización.
PK	organizer_id	INTEGER	No	ID de la organización.
	email	VARCHAR(255)	No	Correo electrónico del contacto de la organización.
	telephone	VARCHAR(20)	Sí	Teléfono del contacto.
FOREIGN KEY (organizer_id) REFERENCES Organizer (id) ON DELETE CASCADE				
UNIQUE (email)				

Event				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SERIAL	No	Identificador único autoincremental para el evento.
	name	VARCHAR(255)	No	Nombre del evento.
	start_date	TIMESTAMP	No	Fecha de inicio del evento.
	end_date	TIMESTAMP	No	Fecha de fin del evento.
	schedule	VARCHAR(255)	No	Horarios del evento.
	description	TEXT	No	Descripción del evento.
	price	REAL	No	Precio del evento.
	image	VARCHAR(255)	Si	Imagen del evento.
	event_status	BOOLEAN	No	Estado del evento.
	event_published	BOOLEAN	No	Estado de publicación del evento.
	event_has_sales	BOOLEAN	No	Indica si el evento tiene o no ventas
	comments	BOOLEAN	No	Estado de activación de los comentarios en el evento.
	organizer_id	INTEGER	No	Organizador del evento.
	location_id	INTEGER	No	Recinto del evento.
FOREIGN KEY (organizer_id) REFERENCES Organizer (id)				
FOREIGN KEY (location_id) REFERENCES Location (id)				
INDEX idx_start_end_published ON Event (start_date, end_date, event_published)				

Event_With_Sales				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SERIAL	No	Identificador único autoincremental para el evento con ventas.
	event_id	INTEGER	No	ID del evento asociado.
	capacity	SMALLINT	No	Aforo del evento.
	sales	SMALLINT	No	Número de entradas vendidas. Valor por defecto: 0
	maximum_per_sale	SMALLINT	No	Máximo de entradas vendidas por transacción.
FOREIGN KEY (event_id) REFERENCES Event (id) ON DELETE CASCADE				

Event_Has_Category				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	event_id	INTEGER	No	ID del evento.
PK	category_id	INTEGER	No	ID de la categoría.
FOREIGN KEY (event_id) REFERENCES Event (id) ON DELETE CASCADE				
FOREIGN KEY (category_id) REFERENCES Category (id)				

Event_Change				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	BIGSERIAL	No	Identificador único autoincremental para el cambio del evento.
PK	event_id	INTEGER	No	ID del evento.
	type	VARCHAR(30)	No	Tipo de cambio.
	date	TIMESTAMP	No	Fecha del cambio.
	description	TEXT	No	Descripción del cambio.

FOREIGN KEY (event_id) REFERENCES Event (id) ON DELETE CASCADE
CHECK type IN ('Delayed', 'Cancelled', 'Location Change', 'Price Change', 'Other')

User				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SERIAL	No	ID del evento.
	name	VARCHAR(255)	No	Nombre del usuario.
	surname	VARCHAR(255)	No	Apellidos del usuario.
	email	VARCHAR(255)	No	Correo electrónico del usuario.
	password	VARCHAR(255)	No	Contraseña del usuario.
	roles	VARCHAR(255)	No	Roles del usuario.
UNIQUE (email)				

Event_Favorite				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	event_id	INTEGER	No	ID del evento.
PK	user_id	INTEGER	No	ID del usuario.
FOREIGN KEY (event_id) REFERENCES Event (id) ON DELETE CASCADE				
FOREIGN KEY (user_id) REFERENCES User (id) ON DELETE CASCADE				

Rating				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	event_id	INTEGER	No	ID del evento.
PK	user_id	INTEGER	No	ID del usuario.
	punctuation	SMALLINT	No	Puntuación del usuario.
	comment	TEXT	No	Comentario del usuario.
	published	BOOLEAN	No	Indica si el comentario está publicado o no.
FOREIGN KEY (event_id) REFERENCES Event (id) ON DELETE CASCADE				

FOREIGN KEY (user_id) REFERENCES User (id) ON DELETE CASCADE

Transaction				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	event_id	INTEGER	No	ID del evento.
PK	user_id	INTEGER	No	ID del usuario.
	unit_price	REAL	No	Precio unitario de las entradas adquiridas.
	quantity	SMALLINT	No	Entradas compradas por el usuario.
	reference	VARCHAR(5)	No	Referencia de la compra.
	date	DATE	No	Fecha del evento. El valor por defecto será la fecha actual (CURRENT_DATE).
FOREIGN KEY (event_id) REFERENCES Event (id)				
FOREIGN KEY (user_id) REFERENCES User (id)				
UNIQUE (reference)				

Para el repositorio estadístico:

City_Statistics				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	city_id	INTEGER	No	ID de la ciudad.
	events	INTEGER	No	Número total de eventos en la ciudad.

Location_Statistics				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	location_id	INTEGER	No	ID del recinto.
	events	INTEGER	No	Número total de eventos en el recinto.

Event_Statistics				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	event_id	INTEGER	No	ID del evento.
	ratings_count	INTEGER	No	Número de comentarios en el evento.
	average_rating	REAL	No	Puntuación promedia del evento.
	total_rating	INTEGER	No	Suma de todas las puntuaciones del evento.
	favorites	INTEGER	No	Número de veces que el evento ha sido marcado como favorito.
	occupation	REAL	Sí	Ocupación del evento.

Transaction_Statistics				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	month	SMALLINT	No	Me.
PK	year	SMALLINT	No	Año.
	transactions	INTEGER	No	Número de transacciones.

System_Counters				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	name	VARCHAR(30)	No	Tipo de contador.
	value	INTEGER	No	Valor del contador.
CHECK name IN (‘non_admin_user’, ‘total_payed_events’, ‘total_events’, ‘total_transactions’)				

Top_Commented_Events				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	event_id	INTEGER	No	ID del evento.
	name	VARCHAR(255)	No	Nombre del evento.

Top_Valued_Events				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	event_id	INTEGER	No	ID del evento.
	name	VARCHAR(255)	No	Nombre del evento.

Top_Sold_Events				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	event_id	INTEGER	No	ID del evento.
	name	VARCHAR(255)	No	Nombre del evento.

Top_Event_Locations				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	location_id	INTEGER	No	ID del recinto.
	name	VARCHAR(255)	No	Nombre del recinto.

Top_Event_Cities				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	city_id	INTEGER	No	ID de la ciudad.
	name	VARCHAR(255)	No	Nombre del recinto.

Top_Favorite_Events				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	event_id	INTEGER	No	ID del evento.
	name	VARCHAR(255)	No	Nombre del evento.

Integer_Indicators				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	indicator	SMALLINT	No	Identificador del indicador.
	value	INTEGER	No	Valor del indicador.
CHECK indicator IN (‘1’)				

Percentage_Indicators				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	indicator	SMALLINT	No	Identificador del indicador.
	value	REAL	No	Valor del indicador.
CHECK indicator IN ('1', '2', '3', '4', '5', '6', '7', '8')				

Para los registros:

Log				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	BIGSERIAL	No	Identificador autoincremental de la tabla de registros.
	date	TIMESTAMP	No	Fecha del registro.
	db_user	VARCHAR(255)	No	Usuario de la base de datos del sistema que efectúa la operación.
	procedure_id	INTEGER	Sí	ID del procedimiento relacionado.
	entry_parameters	TEXT	No	Parámetros de entrada.
	result	TEXT	No	Resultado.
FOREIGN KEY (procedure_id) REFERENCES Procedure(id)				

Procedure				
	Campo	**Tipo**	**Nulo**	**Descripción**
PK	id	SERIAL	No	Identificador autoincremental de la tabla de procedimientos.
	name	VARCHAR(255)	No	Nombre del procedimiento.
	description	VARCHAR(255)	No	Descripción del procedimiento.

4. Implementación de la base de datos

4.1. Configuración del entorno de desarrollo

Para la instalación de la base de datos en local hemos decidido hacer uso de contenedores con Docker.

Podemos definir un contenedor como una unidad estándar de software que contiene código y todas sus dependencias, de manera que la aplicación puede ejecutarse de forma independiente. Un contenedor, por tanto, actúa como una máquina virtual, compartiendo el sistema operativo de la máquina que virtualiza, pero gestionando sus propios recursos.

Docker, por otra parte, es una plataforma de código abierto que permite crear, desplegar y administrar contenedores en un mismo sistema operativo.

Para el desarrollo en local y las pruebas de nuestra base de datos ejecutaremos un contenedor de Docker en el que funcione PostgreSQL.

A la hora de definir nuestro contenedor y su configuración, crearemos un fichero YML que será interpretado por Docker Compose, una herramienta de la plataforma que permite orquestar nuestros contenedores de Docker de manera local, y que creará e iniciará los contenedores con la configuración que hayamos indicado.

Algunos de los principales motivos por la que nos hemos decantado por el uso de Docker son los siguientes:

- No hay necesidad de instalar en el propio sistema operativo el SGBD.
- En caso de necesitar trabajar con un ordenador diferente (por desplazamiento, o en caso de rotura del ordenador habitual de trabajo) simplemente será necesario tener instalado Docker y se pondrá en marcha el mismo entorno de trabajo en las mismas condiciones que disponemos en el ordenador habitual de trabajo.
- En caso de fallos en la base de datos o su configuración, se pueden borrar y reiniciar o borrar fácilmente los contenedores y los volúmenes (capa de persistencia) asociados a ello.

Además, para efectuar todo el desarrollo, usaremos DataGrip, un IDE diseñado por JetBrains para trabajar con bases de datos y SQL.

4.1.2. Git

Para mantener un control de los cambios realizados y evitar las pérdidas de datos se ha decidido usar Git, un software de control de versiones.

El código, así como las actualizaciones que hagamos, se almacenarán en un espacio que se conoce como “repositorio”.

Hemos decidido dar de alta un repositorio en la plataforma GitHub en el que subiremos las diferentes actualizaciones de código que vayamos haciendo. El repositorio se mantendrá como privado durante el desarrollo del trabajo y se hará público una vez se haya presentado.

El repositorio es accesible a través de la dirección https://github.com/danielpilot/event-database

4.1.2.1. Workflow

GIT funciona con un sistema de ramas a las cuales se van subiendo las diferentes actualizaciones de código que vamos haciendo. De esta forma, podremos crear diferentes "ramas" que nos permitirán desarrollar nuestro producto en un área contenida del repositorio.

Los cambios que vayamos haciendo se subirán a las ramas usando confirmaciones (más conocidas como "commits"), que son agrupaciones de los cambios realizados.

Para facilitar la organización de nuestro trabajo, teniendo en cuenta las características de nuestro proyecto, hemos decidido definir un flujo de trabajo con las siguientes características:

- Tenemos permanentemente definidas las siguientes ramas:
  - **main**: es nuestra rama principal, en ella tendremos las versiones más estables de nuestro código.
  - **develop**: en esta rama iremos subiendo las diferentes funcionalidades que hayamos probado.
- Por otra parte, iremos usando diferentes ramas para ir trabajando en nuestras funcionalidades:
  - **Ramas "feature"**: son las ramas que añaden o modifican nuevas funcionalidades. Las creamos a partir de la rama develop y, cuando su contenido esté listo, las mergearemos (es decir, añadiremos su código) a la rama develop.
  - **Ramas "release"**: cuando tengamos un grupo de funcionalidades hecho y probado, crearemos una rama reléase a partir de develop y, tras ello, la mergearemos a main.
- Una vez hayamos añadido contenido a la rama main, crearemos una etiqueta con el formato vX.Y.Z en el que indicaremos la versión con la que estamos trabajando.
  - X indica cambios mayores. En un contexto general de desarrollo de código se indican cambios en los que la funcionalidad es modificada. Para nuestro caso, hemos decidido que durante todo nuestro desarrollo mantendremos una versión mayor 0 (indicando que no es un desarrollo estable) y en cuanto se publique este trabajo final de grado se actualizará a la versión 1.
  - Y indica cambios menores. Se actualiza cada vez que se suba una funcionalidad.

- Z indica parches que resuelvan problemas de las versiones anteriores. En este proyecto no se dará el caso y por tanto asumimos que siempre será 0.

4.2. Espacio virtual

Como hemos visto, cuando trabajamos con bases de datos, existe un nivel lógico y un nivel físico en los que almacenamos los datos. A veces, es necesaria la existencia de un nivel virtual que proporcione independencia entre los niveles lógicos y físicos (por ejemplo, en el caso de bases de datos grandes en las que interese guardar los datos físicos en diferentes volúmenes).

Aunque en este caso concreto no debería ser necesario el uso de espacios virtuales, hemos decidido incorporar y configurar dos diferentes espacios virtuales, asociados a dos rutas físicas diferentes, para ejemplificar las diferentes posibles ubicaciones de la propia base de datos de eventos y del repositorio estadístico (como puede ser un data warehouse).

De esta manera, definimos los siguientes espacios virtuales:

Espacio virtual	Descripción
operational_tablespace	Contiene la base de datos de eventos y el log de acciones.
warehouse_tablespace	Contiene la base de datos del repositorio analítico.

4.3. Usuarios

Para acceder al sistema, definimos los siguientes usuarios:

Usuario	Contraseña	Descripción
postgres	root_event_password	Se trata del super administrador de la base de datos. Puede efectuar todo tipo de acciones sobre todas las bases de datos y usuarios del sistema
event_user	event_user_pass	Todos los permisos sobre el esquema de eventos.
log_user	log_user_pass	Todos los permisos sobre el esquema de registros.
statistics_user	statistics_user_pass	Todos los permisos sobre el repositorio estadístico.

4.4. Instalación e inicialización de la base de datos

Una vez definidos los conceptos y el procedimiento de creación de nuestras bases de datos, comenzaremos implementación de nuestra base de datos.

Crearemos un fichero docker-compose.yml en el que incluiremos la configuración de los datos que queremos cargar:

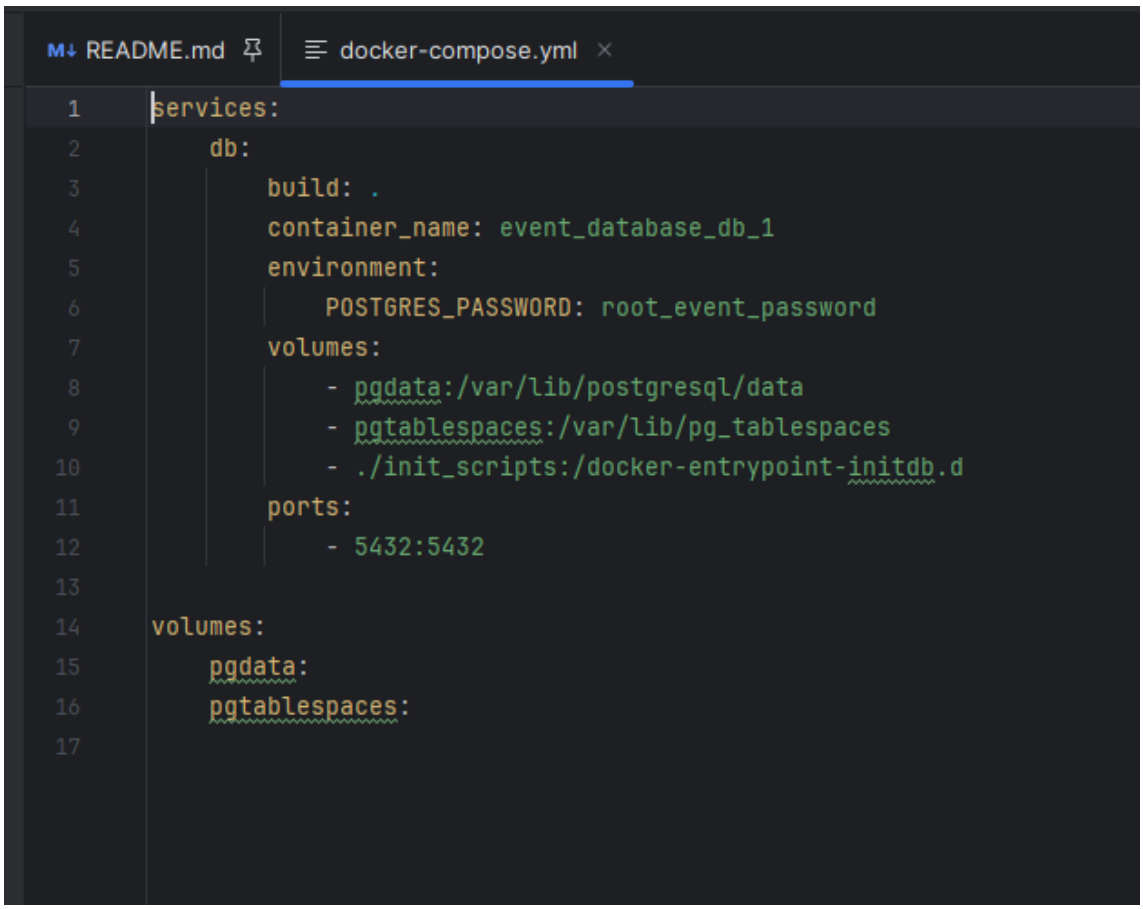


Figura 5: Definición del fichero de Docker

Las configuraciones que hemos seleccionado tienen los siguientes significados:

- **Services:** encontramos un listado de los servicios que se deberán cargar en Docker, cada uno de los cuales se ejecutará en un contenedor diferente, y sus configuraciones. En este caso, únicamente tendremos un servicio (la base de datos) que ejecutará Postgre.
    - **Build**: indica la dirección de una carpeta que contiene un fichero Dockerfile en el que se encuentran las instrucciones de ejecución para construir el contenedor. En este caso, y como únicamente tendremos un contenedor de Docker, lo hemos situado en el directorio raíz.

- **Container_name**: indica el nombre que se pondrá al contenedor y que permitirá identificarlo.

- **Environment**: se indican una serie de variables de entorno que permiten configurar nuestra base de datos. Las posibles variables de entorno que usemos y su funcionamiento vienen referenciadas en la documentación de la imagen de PostgreSQL que implementaremos [16].

- **Volumes**: los volúmenes nos permiten almacenar en una capa de persistencia los cambios que hagamos. Es decir, si no usáramos volúmenes, los datos que almacenáramos en nuestra base de datos se borrarían cada vez que el contenedor se apagara.

  Como podemos ver, usamos dos volúmenes, que también definimos en la sección inferior de volumes (pgdata y pgtablespace). En ellos almacenamos los datos presentes en las dos carpetas del sistema que especificamos: la primera carpeta incluye los datos y espacios virtuales que usa PostgreSQL por defecto, y el segundo volumen incluye los datos físicos que almacenamos en los espacios virtuales que hemos definido.

  Por último, encontramos un volumen especial que llama a /docker-entrypoint-initdb.d. Se trata de una carpeta init_scripts que contiene una serie de ficheros SQL que contienen los scripts que hemos creado para la inicialización de nuestra base de datos: creación de los espacios virtuales, usuarios, bases de datos, tablas, asignación de sus permisos… Estos ficheros se ejecutarán de forma automática y en orden alfabético, por lo que en los diferentes ficheros que creamos hemos incluido un índice numérico al principio de ellos para indicar el orden de ejecución.

- **Ports**: permite exponer los puertos que indiquemos para que puedan ser accesibles de forma externa. Cada uno de los valores especificados en los puertos consiste en dos números de puertos separados por dos puntos (:). El puerto especificado a la izquierda indica el puerto expuesto al exterior, mientras que el puerto especificado a la derecha indica el puerto que se quiere exponer al exterior.

  En este caso, hemos indicado que queremos exponer en el puerto 5432 el contenido del puerto 5432, que es el puerto por defecto de PostgreSQL.

- **Volumes**: indica los volúmenes que Docker creará para almacenar los datos con los que trabajemos. Ambos volúmenes ya han sido mapeados en la sección “volumes” del servicio de bases de datos que hemos definido.

A continuación, exploramos el contenido del fichero Dockerfile que hemos creado para indicar las instrucciones de ejecución de nuestro contenedor:

Figura 6: Contenido del fichero Dockerfile

La primera sentencia, FROM, cargamos la versión 14 de PostgreSQL. Una imagen es una plantilla de solo lectura que define su contenedor y contiene todo el código que se ejecutará. En este caso, se descarga desde el Marketplace “Docker Hub” e instala la aplicación PostgreSQL en el contenedor de Docker que creamos.

Tras ello indicamos una serie de ejecuciones que necesitamos para poder crear los diferentes espacios físicos en los que almacenaremos los datos de nuestras bases de datos (que quedarán mapeados a dichos espacios físicos a través de los espacios virtuales).

Las dos primeras sentencias crean las carpetas en las que se almacenarán los datos y simulan las dos ubicaciones diferentes de nuestra base de datos de eventos y el repositorio estadístico. La tercera sentencia modifica la propiedad y permisos de los directorios para que PostgreSQL pueda escribir en ellos.

4.5. Alta, baja y modificación de datos de eventos

En este apartado, incluiremos toda la información sobre los desarrollos realizados para el alta, baja y modificación de los datos de eventos y el registro de acciones en el log.

Para ello, daremos de alta una serie de diferentes procedimientos y además configuraremos disparadores que nos permitan implementar toda la lógica de negocio especificada en la base de datos.

4.5.1. Tipos

Para facilitar la realización de procedimientos, hemos creado los siguientes tipos con datos:

Event_Sales_Data

Descripción	Contiene datos de las ventas relacionadas con un evento.
Propiedades	• **capacity** (SMALLINT): Aforo del evento. • **maximum_per_sale** (SMALLINT): Número máximo de entradas vendidas por transacción.

4.5.2. Disparadores

Podemos definir los disparadores como unos componentes que se ejecutan automáticamente al producirse un evento determinado.

El uso de disparadores puede sernos de gran utilidad para la implementación de nuestra lógica de negocio.

trg_check_transaction_conditions_before_insert	
Descripción	Comprueba si las condiciones de creación de una nueva transacción se cumplen antes del insertado.
Momento y operación	BEFORE INSERT
Tabla	Transaction
Operativa	Se comprueban los datos del evento relacionado. Calculamos el aforo restante y, en caso de que la venta lo supere, devolvemos un error y bloqueamos la transacción. Recuperamos si la venta está activa para el evento, en caso de que no lo esté, devolvemos un error y bloqueamos la transacción.

trg_check_transaction_conditions_before_update	
Descripción	Comprueba si las condiciones de actualización de una transacción se cumplen.
Momento y operación	BEFORE UPDATE
Tabla	Transaction
Operativa	Se comprueban los datos del evento relacionado. Calculamos el aforo restante y comprobamos que se puede realizar la variación de aforo. En caso de que no se pueda devolvemos un error y bloqueamos la transacción.

	Recuperamos si la venta está activa para el evento, en caso de que no lo esté, devolvemos un error y bloqueamos la transacción.

trg_update_sales_after_insert	
Descripción	Actualiza el número de ventas de un evento tras la creación de la transacción.
Momento y operación	AFTER INSERT
Tabla	Transaction
Operativa	Se actualiza la cantidad de entradas vendidas del evento en la tabla.

trg_update_sales_after_update	
Descripción	Actualiza el número de ventas de un evento tras la actualización de la transacción.
Momento y operación	AFTER UPDATE
Tabla	Transaction
Operativa	Se actualiza la cantidad de entradas vendidas del evento en la tabla.

trg_update_sales_after_delete	
Descripción	Actualiza el número de ventas de un evento tras la eliminación de la transacción.
Momento y operación	AFTER DELETE
Tabla	Transaction
Operativa	Se actualiza la cantidad de entradas vendidas del evento en la tabla.

trg_check_event_sales_status	
Descripción	Comprueba si se dan las condiciones para que el evento tenga activas las ventas y las desactiva si fuera necesario.
Momento y operación	AFTER INSERT

	AFTER UPDATE
Tabla	Event
Operativa	Si el evento ya tiene las ventas desactivadas, no se efectúa ninguna acción. Si el evento no está publicado o está cancelado, cancela la venta del evento.

trg_add_event_change_on_event_modification	
Descripción	Añade automáticamente un cambio de evento cuando se ha producido una modificación de éste.
Momento y operación	AFTER UPDATE
Tabla	Event
Operativa	Todas las inserciones en cambios de eventos se hacen usando un procedimiento creado a tal efecto que se ha definido en la siguiente sección. Si la fecha del evento cambia, se añade un registro que indica si se adelanta o se retrasa su fecha. Si el recinto cambia, se almacena el cambio. Si el precio cambia, se almacena el cambio. Si el evento se cancela, se almacena el cambio.

trg_check_rating_conditions_before_insert_or_update	
Descripción	Comprueba si un comentario se puede añadir o actualizar en función de si están activos los comentarios en el evento.
Momento y operación	BEFORE INSERT BEFORE UPDATE
Tabla	Rating
Operativa	Si el evento tiene los comentarios desactivados, no acepta nuevas valoraciones. Si el evento tiene los cambios desactivados, no acepta nuevas actualizaciones.

4.5.3. Procedimientos

Para la interacción con la base de datos de eventos, hemos decidido crear diferentes procedimientos que nos ayuden a llevar un control y registro de las acciones realizadas en el sistema.

Hemos definido los siguientes procedimientos:

Create_Location	
Descripción	Permite crear un recinto.
Parámetros de entrada	• **_name** (String): Nombre del recinto. • **_address** (String): Dirección del recinto. • **_city_id** (Integer): ID de la ciudad en la que se encuentra el recinto. • **_latitude** (Real): Latitud de la ubicación del recinto. • **_longitude** (Real): Longitud de la ubicación del recinto.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se insertan los valores del nuevo recinto en la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si la ciudad referenciada no existe.
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si se produce cualquier otro error en la operativa.

Update_Location	
Descripción	Permite actualizar un recinto.
Parámetros de entrada	• **_id** (Integer): Identificador del recinto. • **_name** (String): Nombre del recinto. • **_address** (String): Dirección del recinto. • **_city_id** (Integer): ID de la ciudad en la que se encuentra el recinto. • **_latitude** (Real): Latitud de la ubicación del recinto.

	• **_longitude** (Real): Longitud de la ubicación del recinto.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se actualizan los valores del recinto con los nuevos datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si la ciudad referenciada no existe.
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si el identificador suministrado no se corresponde con un valor en la base de datos.
	  ○ Si se produce cualquier otro error en la operativa.

Delete_Location	
Descripción	Permite eliminar un recinto.
Parámetros de entrada	• **_id** (Integer): Identificador del recinto.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina el recinto de la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ S el recinto tiene eventos relacionados.
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si el identificador suministrado no se corresponde con un valor en la base de datos.
	  ○ Si se produce cualquier otro error en la operativa.

Create_Organizer	
Descripción	Permite crear un organizador.
Parámetros de entrada	• **_name** (String): Nombre del organizador. • **_email** (String): Correo electrónico del organizador. • **_type** (String): Tipo de organizador.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se añade el organizador a la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el correo electrónico ya se corresponde con otro resultado en la base de datos.
	  ○ Si el tipo de organizador introducido no es correcto.
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si se produce cualquier otro error en la operativa.

Update_Organizer	
Descripción	Permite actualizar un organizador.
Parámetros de entrada	• **_id** (Integer): Identificador del organizador. • **_name** (String): Nombre del organizador. • **_email** (String): Correo electrónico del organizador. • **_type** (String): Tipo de organizador.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se actualiza el organizador en la base de datos. • Se añade un valor en el registro con el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el correo electrónico ya se

	corresponde con otro resultado en la base de datos. ○ Si el tipo de organizador introducido no es correcto. ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si el identificador suministrado no se corresponde con un valor en la base de datos. ○ Si se produce cualquier otro error en la operativa.

Delete_Organizer	
Descripción	Permite eliminar un organizador.
Parámetros de entrada	• **_id** (Integer): Identificador del organizador.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina el organizador de la base de datos. • Se añade un valor en el registro con el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si el identificador suministrado no se corresponde con un valor en la base de datos. ○ Si el organizador tiene eventos relacionados. ○ Si se produce cualquier otro error en la operativa.

Create_Organizer_Contact	
Descripción	Permite crear un contacto de un organizador.
Parámetros de entrada	• **_organizer_id** (Integer): Identificador del organizador. • **_name** (String): Nombre del contacto de la organización. • **_email** (String): Correo electrónico del contacto de la organización.

	• **_telephone** (String|Null): Teléfono del contacto de la organización.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se comprueba si ya existe otro contacto de organizador con el mismo correo electrónico. • Se añade el contacto del organizador a la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si el correo electrónico ya se corresponde con otro resultado en la base de datos.
	  ○ Si ya existe un contacto de organizador con el mismo nombre asociado al mismo organizador.
	  ○ Si el identificador de organizador introducido no se corresponde con un organizador.
	  ○ Si se produce cualquier otro error en la operativa.

Update_Organizer_Contact	
Descripción	Permite actualizar un contacto de un organizador.
Parámetros de entrada	• **_organizer_id** (Integer): Identificador del organizador. • **_name** (String): Nombre del contacto de la organización. • **_email** (String): Correo electrónico del contacto de la organización. • **_telephone** (String|Null): Teléfono del contacto de la organización.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se actualiza el contacto del organizador a la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el correo electrónico ya se corresponde con otro resultado en la base de datos.
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si no existe un contacto con el ID de organizador y nombre introducido.
	  ○ Si se produce cualquier otro error en la operativa.

Delete_Organizer_Contact	
Descripción	Permite actualizar un contacto de un organizador.
Parámetros de entrada	• **_organizer_id** (Integer): Identificador del organizador. • **_name** (String): Nombre del contacto de la organización.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina el contacto del organizador de la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si no existe un contacto con el ID de organizador y nombre introducido.
	  ○ Si se produce cualquier otro error en la operativa.

Create_Category	
Descripción	Permite crear una categoría.
Parámetros de entrada	• **_name** (String): Nombre de la categoría. • **_parent_id** (Integer|Null): Identificador de la categoría padre en caso de que la tenga.
Operativa	• Se comprueba que el procedimiento está

	dado de alta en la base de datos. • Se añade la categoría a la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si ya existe una categoría con ese nombre en la misma categoría padre.
	  ○ Si, en caso de estar especificada una categoría padre, no se encuentra una categoría con el identificador suministrado.
	  ○ Si se produce cualquier otro error en la operativa.

Update_Category	
Descripción	Permite actualizar una categoría.
Parámetros de entrada	• **_id** (Integer): Identificador de la categoría. • **_name** (String): Nombre de la categoría. • **_parent_id** (Integer|Null): Identificador de la categoría padre en caso de que la tenga.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se comprueba si el ID de la categoría padre introducido es el mismo que el ID de la categoría actual. • Se actualiza la categoría en la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si ya existe una categoría con ese nombre en la misma categoría padre.
	  ○ Si, en caso de estar especificada una categoría padre, no se encuentra una categoría con el identificador suministrado.

	○ Si se especifica que la categoría padre es la propia categoría que actualizamos. ○ Si no se encuentra la categoría a actualizar en la base de datos. ○ Si se produce cualquier otro error en la operativa.

Delete_Category	
Descripción	Permite eliminar una categoría.
Parámetros de entrada	• **_id** (Integer): Identificador de la categoría.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina la categoría de la base de datos. • Se añade un valor en el registro con el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si la categoría tiene eventos relacionados. ○ Si no se encuentra la categoría a actualizar en la base de datos. ○ Si se produce cualquier otro error en la operativa.

Create_User	
Descripción	Permite crear un usuario.
Parámetros de entrada	• **_name** (String): Nombre del usuario. • **_surname** (String): Apellidos del usuario. • **_email** (String): Correo electrónico del usuario. • **_password** (String): Contraseña del usuario. • **_roles** (String): Roles del usuario.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se añade el usuario en la base de datos. • Se añade un valor en el registro con el

	resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si ya existe un usuario con ese correo electrónico.
	  ○ Si se produce cualquier otro error en la operativa.

Update_User	
Descripción	Permite actualizar un usuario.
Parámetros de entrada	• **_id** (Integer): Identificador del usuario. • **_name** (String): Nombre del usuario. • **_surname** (String): Apellidos del usuario. • **_email** (String): Correo electrónico del usuario. • **_password** (String): Contraseña del usuario. • **_roles** (String): Roles del usuario.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se actualiza el usuario en la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si ya existe un usuario con ese correo electrónico.
	  ○ Si no existe el usuario con el identificador dado.
	  ○ Si se produce cualquier otro error en la operativa.

Delete_User	
Descripción	Permite actualizar un usuario.
Parámetros de entrada	• **_id** (Integer): Identificador del usuario.

Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina el usuario de la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si el usuario tiene transacciones asociadas.
	  ○ Si no existe el usuario con el identificador dado.
	  ○ Si se produce cualquier otro error en la operativa.

Create_Event	
Descripción	Permite crear un evento.
Parámetros de entrada	• **_name** (String): Nombre del evento. • **_start_date** (Date): Fecha de inicio del evento. • **_end_date** (Date): Fecha de fin del evento. • **_schedule** (String): Horario del evento. • **_description** (String): Descripción del evento. • **_price** (Float): Precio del evento. • **_image** (String|Null): Imagen del evento. • **_event_status** (Boolean): Estado de planificación del evento. • **_event_published** (Boolean): Estado de publicación del evento. • **_comments** (Boolean): Estado de comentarios del evento. • **_organizer_id** (Integer): Identificador del organizador del evento. • **_location_id** (Integer): Identificador del recinto del evento. • **_categories** (Integer[]|Null): Listado de identificadores de las categorías asociadas al avento. • **_event_sales_data** (EventSalesData|Null): Datos de venta del evento.
Operativa	• Se comprueba que el procedimiento está

	dado de alta en la base de datos. • Comprobamos que, en caso de suministrarse categorías, todas las categorías existen. • Si no se han pasado datos de venta del evento, ajustamos el valor de la variable event_has_sales a falso. En caso contrario, será verdadero. • Se añade el evento a la base de datos. • Se añade cada una de las categorías a la tabla que relaciona los eventos con la categoría (Event_Has_Category). • Si _event_sales_data no es nulo, se crea una nueva entidad Event_With_Sales que se relaciona con el evento. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si hay categorías que no existen.
	  ○ Si se produce cualquier otro error en la operativa.

Update_Event	
Descripción	Permite actualizar un evento.
Parámetros de entrada	• **_id** (Integer): Identificador del evento a actualizar. • **_name** (String): Nombre del evento. • **_start_date** (Date): Fecha de inicio del evento. • **_end_date** (Date): Fecha de fin del evento. • **_schedule** (String): Horario del evento. • **_description** (String): Descripción del evento. • **_price** (Float): Precio del evento. • **_image** (String|Null): Imagen del evento. • **_event_status** (Boolean): Estado de planificación del evento. • **_event_published** (Boolean): Estado de publicación del evento. • **_comments** (Boolean): Estado de comentarios del evento.

	• **_organizer_id** (Integer): Identificador del organizador del evento. • **_location_id** (Integer): Identificador del recinto del evento. • **_categories** (Integer[]|Null): Listado de identificadores de las categorías asociadas al avento. • **_event_sales_data** (EventSalesData|Null): Datos de venta del evento.

Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos.
	• Comprobamos que, en caso de suministrarse categorías, todas las categorías existen.
	• Si no se han pasado datos de venta del evento, ajustamos el valor de la variable event_has_sales a falso. En caso contrario, será verdadero.
	• Se actualiza el evento en la base de datos.
	• Se eliminan todas las categorías asociadas por el evento y, tras ello, se añade cada una de las categorías a la tabla que relaciona los eventos con la categoría.
	• Si _event_sales_data no es nulo, se pueden dar dos casos distintos:
	  ○ Si ya existe una entidad Event_With_Sales asociada, se actualizan sus datos.
	  ○ Si no existe una entidad Event_With_Sales asociada, se crea dicha entidad y se asocia al evento.
	• Si _event_sales_data es nulo, intenta borrar el Event_With_Sales relacionado.
	  ○ Si no tiene transacciones relacionada, lo borra de la base de datos.
	  ○ Si tiene transacciones relacionadas, no lo borra pero lanza un aviso. Sin embargo, si que actualiza todo el resto de datos.
	• Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si hay categorías que no existen.

	○ Si no se encuentra el evento con el identificador suministrado. ○ Si no quedan suficientes entradas disponibles. ○ Si la venta de entradas para el evento está cerrada. ○ Si se produce cualquier otro error en la operativa. • **INFO**: ○ Si el Evento con ventas tiene transacciones relacionadas: En ese caso, se conservará la entidad Event_With_Sales y se desactivarán sus ventas pero se mostrará un mensaje de error.

Delete_Event	
Descripción	Permite eliminar un evento.
Parámetros de entrada	• **_id** (Integer): Identificador del evento a eliminar.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina el evento de la base de datos. • Se añade un valor en el registro con el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si no se encuentra el evento con el identificador suministrado. ○ Si el Evento con ventas tiene transacciones relacionadas. ○ Si se produce cualquier otro error en la operativa

Create_Event_Change	
Descripción	Permite crear un registro de cambio en el evento.
Parámetros de entrada	• **_name_id** (Integer): Identificador del evento. • **_type** (String): Tipo de cambio. • **_description** (String): Descripción del

	cambio.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se añade el cambio a la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si el identificador del evento suministrado no se corresponde con un evento en la base de datos.
	  ○ Si el tipo de cambio suministrado no se corresponde con un tipo válido.
	  ○ Si se produce cualquier otro error en la operativa.

Update_Event_Change	
Descripción	Permite actualizar un registro de cambio en el evento.
Parámetros de entrada	• **_id** (Integer): Identificador del cambio. • **_name_id** (Integer): Identificador del evento. • **_type** (String): Tipo de cambio. • **_description** (String): Descripción del cambio.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se actualiza el cambio a la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si el identificador del evento suministrado no se corresponde con un evento en la base de datos.
	  ○ Si el tipo de cambio suministrado no se corresponde con un tipo válido.
	  ○ Si el identificador del cambio no se

	corresponde con un cambio en la base de datos. ○ Si se produce cualquier otro error en la operativa.

Delete_Event_Change	
Descripción	Permite eliminar un registro de cambio en el evento.
Parámetros de entrada	• **_id** (Integer): Identificador del cambio.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina el cambio de la base de datos. • Se añade un valor en el registro con el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si el identificador del cambio no se corresponde con un cambio en la base de datos. ○ Si se produce cualquier otro error en la operativa.

Create_Rating	
Descripción	Permite crear un comentario del usuario.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento. • **_user_id** (Integer): Identificador del usuario. • **_punctuation** (Integer): Puntuación otorgada al evento. • **_comment** (String): Comentario del evento. • **_published** (Boolean): Indica si el comentario está puntuado.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se comprueba que la puntuación no es mayor que 5 y menor que 0. • Se comprueba que el evento existe. • Se comprueba que el usuario existe. • Se añade el comentario a la base de datos.

	• Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si el identificador del evento suministrado no se corresponde con un evento en la base de datos.
	  ○ Si el identificador del usuario suministrado no se corresponde con un usuario en la base de datos.
	  ○ Si ya existe un comentario del usuario para ese evento.
	  ○ Si se introduce una puntuación mayor que 5 o menor que 0.
	  ○ Si los comentarios del evento no están activados.
	  ○ Si se produce cualquier otro error en la operativa.

Update_Rating	
Descripción	Permite crear un comentario del usuario.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento. • **_user_id** (Integer): Identificador del usuario. • **_punctuation** (Integer): Puntuación otorgada al evento. • **_comment** (String): Comentario del evento. • **_published** (Boolean): Indica si el comentario está puntuado.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se comprueba que la puntuación no es mayor que 5 ni menor que 0. • Se comprueba que el evento existe. • Se comprueba que el usuario existe. • Se actualiza el comentario a la base de datos. • Se añade un valor en el registro con el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente.

	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si el identificador del evento suministrado no se corresponde con un evento en la base de datos.
	  ○ Si el identificador del usuario suministrado no se corresponde con un usuario en la base de datos.
	  ○ Si se introduce una puntuación mayor que 5 o menos que 0.
	  ○ Si no se encuentra el comentario en la base de datos.
	  ○ Si los comentarios del evento no están activados.
	  ○ Si se produce cualquier otro error en la operativa.

Delete_Rating	
Descripción	Permite eliminar un comentario del usuario.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento. • **_user_id** (Integer): Identificador del usuario.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina el comentario a la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si no se encuentra el comentario en la base de datos.
	  ○ Si se produce cualquier otro error en la operativa

Create_Transaction	
Descripción	Permite crear una transacción.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento (a través de Event_With_Sales). • **_user_id** (Integer): Identificador del usuario.

	• **_unit_price** (Float): Precio unitario de las entradas. • **__quantity** (Float): Cantidad de entradas compradas. • **_reference** (String): Referencia única de la transacción.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se obtiene si el evento tiene el sistema de ventas activado y, en caso de tenerlo, se recupera el número de entradas máximas que se puede devolver por usuario. • Se comprueba si el número de entradas adquirido es mayor que el número de entradas máximo vendido por usuario. • Se comprueba que el usuario existe. • Se comprueba que no existe una referencia igual. • Se añade la transacción a la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si el identificador del evento suministrado no se corresponde con un evento en la base de datos.
	  ○ Si el identificador del usuario suministrado no se corresponde con un usuario en la base de datos.
	  ○ El sistema de ventas no está activo para el evento.
	  ○ Se intentan comprar más entradas de las disponibles.
	  ○ Si ya existe una transacción con la referencia introducida.
	  ○ Si ya existe una transacción para ese usuario y evento.
	  ○ Si las ventas del evento están cerradas.
	  ○ Si la venta de entradas supera el aforo del evento.
	  ○ Si se produce cualquier otro error en la operativa.

Update_Transaction	
Descripción	Permite actualizar una transacción.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento (a través de Event_With_Sales). • **_user_id** (Integer): Identificador del usuario. • **_unit_price** (Float): Precio unitario de las entradas. • **__quantity** (Float): Cantidad de entradas compradas. • **_reference** (String): Referencia única de la transacción.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se obtiene si el evento tiene el sistema de ventas activado y, en caso de tenerlo, se recupera el número de entradas máximas que se puede devolver por usuario. • Se comprueba si el número de entradas adquirido es mayor que el número de entradas máximo vendido por usuario. • Se comprueba que el usuario existe. • Se actualiza la transacción en la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si el identificador del evento suministrado no se corresponde con un evento en la base de datos.
	  ○ Si el identificador del usuario suministrado no se corresponde con un usuario en la base de datos.
	  ○ El sistema de ventas no está activo para el evento.
	  ○ Se intentan comprar más entradas de las disponibles.
	  ○ Si ya existe una transacción con la referencia introducida.
	  ○ Si no se encuentra la transacción para el usuario y evento introducido.
	  ○ Si se produce cualquier otro error en

	la operativa.

Delete_Transaction	
Descripción	Permite eliminar una transacción.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento (a través de Event_With_Sales). • **_user_id** (Integer): Identificador del usuario.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina la transacción de la base de datos. • Se añade un valor en el registro con el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si no se encuentra la transacción para el usuario y evento introducido.
	  ○ Si se produce cualquier otro error en la operativa.

Create_Event_Favorite	
Descripción	Permite añadir un favorito a un evento.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento. • **_user_id** (Integer): Identificador del usuario.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se comprueba que el evento existe. • Se comprueba que el usuario existe. • Se añade el favorito a la base de datos. • Se añade el valor en el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si no se encuentra el evento en la

	base de datos. ○ Si no se encuentra el usuario en la base de datos. ○ Si el usuario ya tiene el evento como favorito. ○ Si se produce cualquier otro error en la operativa.

Delete_Event_Favorite	
Descripción	Permite actualizar un favorito a un evento.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento. • **_user_id** (Integer): Identificador del usuario.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina el favorito a la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si no se encuentra el favorito en la base de datos. ○ Si se produce cualquier otro error en la operativa.

Create_Country	
Descripción	Permite añadir un país.
Parámetros de entrada	• **_name** (String): Nombre del país.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se añade el país a la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos.

	○ Existe un país con el mismo nombre. ○ Si se produce cualquier otro error en la operativa.

Update_Country	
Descripción	Permite añadir un país.
Parámetros de entrada	• **_id** (Integer): Identificador del país. • **_name** (String): Nombre del país.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se añade el país a la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ No existe un país con el identificador dado. ○ Existe un país con el mismo nombre. ○ Si se produce cualquier otro error en la operativa.

Delete_Country	
Descripción	Permite eliminar un país.
Parámetros de entrada	• **_id** (Integer): Identificador del país.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina el país de la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ No existe un país con el identificador dado. ○ El país tiene regiones asociadas.

	○ Si se produce cualquier otro error en la operativa.

Create_Region	
Descripción	Permite añadir una región.
Parámetros de entrada	• **_name** (String): Nombre de la región. • **_country_id** (Integer). Identificador del país.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se añade la región a la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si no existe el país suministrado con el identificador. ○ Si se produce cualquier otro error en la operativa.

Update_Region	
Descripción	Permite actualizar una región.
Parámetros de entrada	• **_id** (Integer): Identificador de la región. • **_name** (String): Nombre de la región. • **_country_id** (Integer). Identificador del país.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se actualiza la región de la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si no existe el país suministrado con el identificador. ○ Si no existe la región con el

	identificador suministrado. ○ Si se produce cualquier otro error en la operativa.

Delete_Region	
Descripción	Permite eliminar una región.
Parámetros de entrada	• **_id** (Integer): Identificador de la región. • **_name** (String): Nombre de la región.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se actualiza la región de la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si no existe la región con el identificador suministrado. ○ Si la región tiene provincias asociadas. ○ Si se produce cualquier otro error en la operativa.

Create_Province	
Descripción	Permite crear una provincia.
Parámetros de entrada	• **_name** (String): Nombre de la provincia. • **_region_id** (Integer). Identificador de la región.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se añade la provincia en la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos.

	○ Si no existe la región con el identificador suministrado. ○ Si se produce cualquier otro error en la operativa.

Update_Province	
Descripción	Permite actualizar una provincia.
Parámetros de entrada	• **_id** (Integer): Identificador de la provincia. • **_name** (String): Nombre de la provincia. • **_region_id** (Integer). Identificador de la región.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se actualiza la provincia en la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si no existe la región con el identificador suministrado. ○ Si no existe la provincia con el identificador suministrado. ○ Si se produce cualquier otro error en la operativa.

Delete_Province	
Descripción	Permite eliminar una provincia.
Parámetros de entrada	• **_id** (Integer): Identificador de la provincia.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina la provincia de la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**:

	○ Si el procedimiento no está dado de alta en la base de datos. ○ Si no existe la provincia con el identificador suministrado. ○ Si la provincia tiene ciudades asociadas. ○ Si se produce cualquier otro error en la operativa.

Create_City	
Descripción	Permite crear una ciudad.
Parámetros de entrada	• **_name** (String): Nombre de la ciudad. • **_province_id** (Integer). Identificador de la provincia.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se añade la ciudad en la base de datos. • Se añade el valor en el resultado de la operativa.
Salida	• **OK**: si el proceso se ha completado correctamente. • **ERROR**: ○ Si el procedimiento no está dado de alta en la base de datos. ○ Si no existe la provincia con el identificador suministrado. ○ Si se produce cualquier otro error en la operativa.

Update_City	
Descripción	Permite actualizar una ciudad.
Parámetros de entrada	• **_id** (Integer): Identificador de la ciudad. • **_name** (String): Nombre de la ciudad. • **_province_id** (Integer). Identificador de la provincia.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se actualiza la ciudad en la base de datos. • Se añade el valor en el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si no existe la provincia con el identificador suministrado.
	  ○ Si no existe la ciudad con el identificador suministrado.
	  ○ Si se produce cualquier otro error en la operativa.

Delete_Province	
Descripción	Permite eliminar una ciudad.
Parámetros de entrada	• **_id** (Integer): Identificador de la ciudad.
Operativa	• Se comprueba que el procedimiento está dado de alta en la base de datos. • Se elimina la ciudad de la base de datos. • Se añade el valor en el resultado de la operativa.

Salida	• **OK**: si el proceso se ha completado correctamente.
	• **ERROR**:
	  ○ Si el procedimiento no está dado de alta en la base de datos.
	  ○ Si no existe la ciudad con el identificador suministrado.
	  ○ Si la ciudad tiene recintos asociados.
	  ○ Si se produce cualquier otro error en la operativa.

4.6. Repositorio estadístico

4.6.1. Procedimientos auxiliares

Para simplificar la lógica de los diferentes disparadores que configuraremos, y dado que ciertas operativas se repiten entre ellos, hemos configurado una serie de procedimientos que pueden ser usados en los diferentes disparadores:

Increase_Statistic_Ratings	
Descripción	Para el caso de un añadido de un nuevo comentario con puntuación, permite actualizar el número de comentarios, la valoración total y la valoración promedio de las estadísticas de un evento.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento. • **_punctuation** (Integer): Puntuación otorgada al evento.
Operativa	• Se comprueba si las estadísticas para el evento en cuestión existen o no en la base de datos. • Si las estadísticas para el evento existen, las actualiza, incrementando en uno el número de comentarios, actualiza la puntuación total del evento y la puntuación promedio. • Si las estadísticas para el evento no existen, las crea.

Decrease_Statistic_Ratings	
Descripción	Para el caso del borrado de un comentario con puntuación, permite actualizar el número de comentarios, la valoración total y la valoración promedio de las estadísticas de un evento.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento. • **_punctuation** (Integer): Puntuación otorgada al evento.
Operativa	• Se comprueba si las estadísticas para el evento en cuestión existen o no en la base de datos. • Si las estadísticas para el evento existen, las actualiza, decrementando en uno el número de comentarios, actualiza la puntuación total del evento y la puntuación promedio.

Increase_Statistic_Favorites	
Descripción	Para el caso de un añadido de un nuevo favorito, permite incrementar en 1 el número de favoritos en las estadísticas de un evento dado.

Parámetros de entrada	• **_event_id** (Integer): Identificador del evento.
Operativa	• Se comprueba si las estadísticas para el evento en cuestión existen o no en la base de datos. • Si las estadísticas para el evento existen, las actualiza, incrementando en uno el número de favoritos. • Si las estadísticas para el evento no existen, las crea.

Decrease_Statistic_Favorites	
Descripción	Para el caso del borrado de un favorito, se decrementa en uno el número de favoritos en las estadísticas de un evento dato.
Parámetros de entrada	• **_event_id** (Integer): Identificador del evento.
Operativa	• Se comprueba si las estadísticas para el evento en cuestión existen o no en la base de datos. • Si las estadísticas para el evento existen, las actualiza, decrementando en uno el número de favoritos.

Increase_Statistic_Location	
Descripción	Permite incrementar el número de eventos en uno en un recito dado.
Parámetros de entrada	• **_location_id** (Integer): Identificador del recinto.
Operativa	• Se comprueba si las estadísticas para el recinto en cuestión existen o no en la base de datos. • Si las estadísticas para el recinto existen, las actualiza, incrementando en uno el número de eventos. • Si las estadísticas para el recinto no existen, las crea. • Incrementa las estadísticas de la ciudad en la que tiene lugar el recinto en uno, llamando al procedimiento Increase_Statistic_City.

Decrease_Statistic_Location	
Descripción	Permite decrementar el número de eventos en uno en un reciento.
Parámetros de entrada	• **_location_id** (Integer): Identificador del recinto.
Operativa	• Se comprueba si las estadísticas para el recinto en cuestión existen o no en la base de datos. • Si las estadísticas para la ciudad existen, las actualiza, decrementando en uno el número de eventos. • Decrementa las estadísticas de la ciudad en la que tiene lugar el recinto en uno, llamando al procedimiento Decrease_Statistic_City.

Increase_Statistic_City	
Descripción	Permite incrementar el número de eventos en uno en una ciudad dada.
Parámetros de entrada	• **_location_id** (Integer): Identificador de la ciudad.
Operativa	• Se comprueba si las estadísticas para la ciudad en cuestión existen o no en la base de datos. • Si las estadísticas para la ciudad existen, las actualiza, incrementando en uno el número de eventos. • Si las estadísticas para la ciudad no existen, las crea.

Decrease_Statistic_City	
Descripción	Permite decrementar el número de eventos en uno en una ciudad dada.
Parámetros de entrada	• **_location_id** (Integer): Identificador de la ciudad.
Operativa	• Se comprueba si las estadísticas para la ciudad en cuestión existen o no en la base de datos. • Si las estadísticas para la ciudad existen, las actualiza, decrementando en uno el número

	de eventos.

Update_Average_Transactions_Per_User	
Descripción	Actualiza el promedio de transacciones por usuario.
Parámetros de entrada	No dispone.
Operativa	• Se calcula el promedio de transacciones por usuario dividiendo el número total de transacciones entre el número total de usuarios no administradores del sistema. El resultado se almacena en la tabla Percentage_Indicators.

Increase_Statistic_Event_Counter	
Descripción	Incrementa el número total de eventos del sistema.
Parámetros de entrada	No dispone.
Operativa	• Incrementa en 1 el número total de eventos del sistema almacenado en System_Counters.

Decrease_Statistic_Event_Counter	
Descripción	Decrementa el número total de eventos del sistema.
Parámetros de entrada	No dispone.
Operativa	• Decrementa en 1 el número total de eventos del sistema almacenado en System_Counters.

Increase_Statistic_Event_Sales_Counter	
Descripción	Incrementa el número total de eventos con venta de entradas del sistema.
Parámetros de entrada	No dispone.
Operativa	• Incrementa en 1 el número total de eventos con venta de entradas del sistema almacenado en System_Counters.

Decrease_Statistic_Event_Sales_Counter	
Descripción	Decrementa el número total de eventos con venta de entradas del sistema.
Parámetros de entrada	No dispone.
Operativa	• Decrementa en 1 el número total de eventos con venta de entradas del sistema almacenado en System_Counters.

Update_Average_Price	
Descripción	Actualiza el precio medio de los eventos con venta de entrada del sistema.
Parámetros de entrada	• **_event_price** (Real): Precio del evento que modifica el precio. • **_is_increment** (Boolean): Indica si la modificación es un incremento o decremento del precio.
Operativa	• En función del valor de "_is_increment" introducido, incrementamos o decrementamos el precio del evento al precio total de los eventos en Percentage_Indicators. • Recupera el número total de eventos de pago del sistema. • Si no hay eventos de pago en el sistema, se actualiza el precio de los eventos 0 y se detiene la ejecución del procedimiento. • Recupera el precio total de los eventos de Percentage_Indicators. • Calcula el precio medio de los eventos del sistema dividiendo la suma de los precios de los eventos entre el número de eventos de pago del sistema, redondeando el resultado a dos resultados, y actualizando el indicador del sistema correspondiente.

Update_Payed_Events_Percentage	
Descripción	Actualiza el porcentaje de eventos de pago en el sistema.
Parámetros de entrada	No dispone.
Operativa	• Recupera el número total de eventos del

	sistema. • Recupera el número total de eventos de pago del sistema. • Actualiza el porcentaje de eventos de pago del sistema si el número total de eventos es 0 y detiene la ejecución del procedimiento. • Calcula y actualiza el indicador correspondiente al porcentaje de eventos de pago en el sistema. Para ello, divide el número de eventos de pago entre el número total de eventos y lo multiplica por 100, redondeando el resultado a dos decimales.

Update_Transaction_Statistics	
Descripción	Actualiza los datos estadísticos de las transacciones del sistema.
Parámetros de entrada	• **_transaction_month** (Integer): mes en el que tiene lugar la transacción. • **_transaction_year** (Integer): año en el que tiene lugar la transacción. • **_is_increment** (Boolean): Indica si debe incrementarse o decrementarse el número de transacciones.

Operativa	• Comprueba si ya existen transacciones o no para el mes y año indicado.
	  ○ Si ya existen transacciones, incrementa o decrementa en 1 el número de transacciones en función del valor de _is_increment indicado.
	  ○ Si no existen transacciones, incrementa en 1 el número de transacciones si "_is_increment" es verdadero.

Update_Transaction_Variation_Last_Month	
Descripción	Actualiza la variación de las estadísticas del mes en curso respecto al mes anterior.
Parámetros de entrada	No dispone.
Operativa	• Recupera el número de transacciones durante el mes en curso. En caso de no disponer, se presupone que son 0. • Recupera el número de transacciones en el

	mes precedente. En caso de no disponer, se presupone que son 0.
	• Calculamos el porcentaje de variación de transacciones en el mes actual respecto al anterior:
	  ○ Si en el mes anterior el número de transacciones es 0, presuponemos que el porcentaje de variación es del 100%.
	  ○ En caso contrario, el porcentaje se calcula de la siguiente manera:
	
	  Δ% = ((V_actual - V_anterior) / V_anterior) * 100
	
	  Donde V_actual es el número de transacciones en el mes actual y V_anterior el número de transacciones en el mes previo.
	
	  Se redondea el resultado a dos decimales.
	• Se actualiza el indicador correspondiente a la variación de transacciones respecto al mes anterior.

Update_Event_With_Sales_Occupation	
Descripción	Actualiza las estadísticas de ocupación de un evento con ventas.
Parámetros de entrada	• **_event_with_sales_id** (Integer): Identificador de los datos de ventas del evento.

Operativa	• Recuperamos el identificador del evento, la capacidad y el número de ventas realizadas en el evento.
	• Calculamos la ocupación actual del evento:
	  ○ Si la capacidad es 0, el porcentaje de ocupación también será 0.
	  ○ Si la capacidad no es 0, calculamos el porcentaje de ocupación dividiendo el número de ventas entre la capacidad del evento y multiplicando por 100. Redondeamos el resultado a 2 decimales.
	• Recuperamos el valor de ocupación del evento almacenado en las estadísticas.

	○ Si no había estadísticas para el evento, las creamos indicando el valor de ocupación del evento. ○ Si las estadísticas del evento ya existían, actualizamos la ocupación del evento. • Si el evento no estaba completo y pasa a estarlo, incrementamos el indicador de eventos completos en 1. • Si el evento estaba completo y deja de estarlo, decrementamos el indicador de eventos completos en 1. • Actualizamos el indicador que mantiene en la base de datos la suma de los porcentajes de ocupación del sistema. Si no existía un porcentaje previo, sumamos el nuevo valor al indicador y en caso contrario sumamos la diferencia entre la ocupación previa y la actual. • Recuperamos de “System_Counters” el número total de eventos de pago. • Recuperamos de “Percentage_Indicators” la suma de porcentajes de ocupación de los eventos. • Actualizamos el porcentaje medio de ocupación de los eventos de pago dividiendo la suma de los porcentajes de ocupación entre el número total de eventos de pago y redondeando el resultado a dos decimales. • Recuperamos el número total de eventos completos. • Actualizamos el porcentaje de eventos de pago completos dividiendo el número total de eventos completos entre el número total de eventos de pago y multiplicando por 100. Redondeamos el resultado a dos decimales.

4.6.2. Disparadores

trg_update_event_statistics_on_rating_insert	
Descripción	Actualiza la tabla de estadísticas de un evento cuando se añade un nuevo comentario con valoración.
Momento y operación	AFTER INSERT

Tabla	Rating
Operativa	Se comprueba si el nuevo comentario que se inserta está o no publicado. En caso de que no esté publicado, la estadística no se tiene en cuenta. En caso de estar publicado, se llama al procedimiento auxiliar Increase_Statistic_Ratings.

trg_update_event_statistics_on_rating_update	
Descripción	Actualiza la tabla de estadísticas de un evento cuando se actualiza un comentario con valoración.
Momento y operación	AFTER UPDATE
Tabla	Rating
Operativa	Comprobamos el estado de publicación del comentario: • Si el comentario no estaba publicado y sigue sin estar publicado, no se efectúan acciones. • Si el comentario no estaba publicado y pasa a estar publicado, añadimos las estadísticas del evento a través del procedimiento Increase_Statistic_Ratings. • Si el comentario estaba publicado y deja de estarlo, a nivel estadístico es como si se hubiera borrado. Se actualizan las estadísticas llamando al procedimiento Decrease_Statistic_Ratings. • Si el comentario estaba publicado y sigue publicado, actualizamos la puntuación media y total de las estadísticas del evento calculando la diferencia de puntuación entre ambas.

trg_update_event_statistics_on_rating_delete	
Descripción	Actualiza la tabla de estadísticas de un evento cuando se elimina un comentario con valoración.
Momento y operación	AFTER DELETE
Tabla	Rating

Operativa	Comprobamos el estado de publicación del comentario: • Si el comentario no estaba publicado, no se modifican las estadísticas. • Si el comentario estaba publicado, se actualizan las estadísticas llamando al procedimiento Decrease_Statistic_Ratings.

trg_update_event_statistics_on_favorite_insert	
Descripción	Actualiza la tabla de estadísticas de un evento cuando es marcado como favorito.
Momento y operación	AFTER INSERT
Tabla	Event_Favorite
Operativa	Se actualizan las estadísticas llamando al procedimiento Increase_Statistic_Favorites.

trg_update_event_statistics_on_favorite_update	
Descripción	Actualiza la tabla de estadísticas de un evento cuando el favorito es actualizado.
Momento y operación	AFTER UPDATE
Tabla	Event_Favorite
Operativa	Se actualizan las estadísticas cuando se produce una actualización en la tabla de favoritos. • Si el ID del evento no varía, no se producen modificaciones. • Si el ID del evento varía, se decrementa el número de favoritos en el evento anterior a través del procedimiento Decrease_Statistic_Favorites y se incrementa en el nuevo usando Increase_Statistic_Favorites.

trg_update_event_statistics_on_favorite_delete	
Descripción	Actualiza la tabla de estadísticas de un evento cuando el favorito es eliminado.
Momento y operación	AFTER DELETE

Tabla	Event_Favorite
Operativa	• Se actualizan las estadísticas llamando al procedimiento Decrease_Statistic_Favorites.

trg_update_event_statistics_on_event_insert	
Descripción	Actualiza las estadísticas cuando se añade un nuevo evento.
Momento y operación	AFTER INSERT
Tabla	Event

Operativa	Se comprueba si el evento está publicado y en estado programado.
	
	• Si el evento no está publicado o en estado programado, no se efectúan acciones.
	• Se actualizan las estadísticas del evento asociadas al recinto y la ciudad llamando al procedimiento Increase_Statistic_Location.
	• Se incrementa el contador de eventos del sistema llamando al procedimiento Increase_Statistic_Event_Counter.
	• Si el evento tiene el sistema de ventas activo:
	  ○ Se incrementa el contador de eventos de pago llamando al procedimiento Increase_Statistic_Event_Sales_Cou nter.
	  ○ Se actualiza el precio medio de los eventos a través del procedimiento Update_Average_Price.
	• Se actualiza el porcentaje de eventos pagados llamando al procedimiento Update_Payed_Events_Percentage.

trg_update_event_statistics_on_event_update	
Descripción	Actualiza las estadísticas cuando se actualiza un evento.
Momento y operación	AFTER UPDATE
Tabla	Event
Operativa	Se actualizan las estadísticas cuando se produce una modificación de un evento:

	• Si el evento pasa de estar de estado no publicado o no programado a publicado:
	  ○ Se incrementan las estadísticas asociadas al recinto y la ciudad llamando al procedimiento Increase_Statistic_Location.
	  ○ Se incrementa el contador de eventos del sistema llamando al procedimiento Increase_Statistic_Event_Counter.
	  ○ Si el evento tiene el sistema de ventas activo, se incrementa el contador de eventos de pago llamando al procedimiento Increase_Statistic_Event_Sales_Counter y se actualiza el precio medio de los eventos a través del procedimiento Update_Average_Price.
	  ○ Se actualiza el porcentaje de eventos pagados llamando al procedimiento Update_Payed_Events_Percentage.
	• Si el evento pasa de estar de estado publicado y programado a estado no publicado o no programado:
	  ○ Se decrementan las estadísticas asociadas al recinto y la ciudad llamando al procedimiento Decrease_Statistic_Location.
	  ○ Se decre,emta el contador de eventos del sistema llamando al procedimiento Decrease_Statistic_Event_Counter.
	  ○ Si el evento tenía el sistema de ventas activo, se decrementa el contador de eventos de pago llamando al procedimiento Decrease_Statistic_Event_Sales_Counter y se actualiza el precio medio de los eventos a través del procedimiento Update_Average_Price.
	  ○ Se actualiza el porcentaje de eventos pagados llamando al procedimiento Update_Payed_Events_Percentage.
	• Comprobamos si se producen modificaciones en un recinto cancelado o no publicado. Si es el caso, no efectuamos acciones adicionales.
	• Era y sigue siendo de pago, se actualiza el precio medio de los eventos llamando al procedimiento Update_Average_Price.

	• Si el evento era de pago y deja de serlo, decrementamos el número de eventos de pago llamando al procedimiento Decrease_Statistic_Event_Sales_Counter y actualizamos el precio medio de los eventos llamando al método Update_Average_Price. • Si un evento no era de pago y pasa a serlo, incrementamos el número de eventos de pago llamando al procedimiento Increase_Statistic_Event_Sales_Counter y actualizamos el precio medio de los eventos llamando al procedimiento Update_Average_Price. • Si el recinto del evento varía, se incrementan las estadísticas del nuevo recinto en el que tiene lugar el evento, llamando al procedimiento Increase_Statistic_Location, y se decrementan las estadísticas del recinto en el que previamente tenía lugar el evento, llamando al procedimiento Decrease_Statistic_Location.

trg_update_event_statistics_on_event_delete	
Descripción	Actualiza las estadísticas cuando se elimina un evento.
Momento y operación	AFTER DELETE
Tabla	Event
Operativa	Se comprueba si el evento estaba publicado y en estado programado. • Si no estaba publicado y programado, no se efectúan modificaciones en las estadísticas. • Si el evento estaba publicado y programado, se actualizan las estadísticas del recinto y ciudad asociadas llamando al procedimiento Decrease_Statistic_Location. • Si el evento tenía el sistema de ventas activo, decrementamos el número de eventos de pago llamando al procedimiento Decrease_Statistic_Event_Sales_Counter y actualizamos el precio medio de los eventos llamando al procedimiento Update_Average_Price. • Actualizamos el porcentaje de eventos de pago del sistema llamando al procedimiento

	Update_Payed_Events_Percentage.

trg_update_user_statistics_on_user_insert	
Descripción	Actualiza las estadísticas de usuario tras la inserción de un usuario.
Momento y operación	AFTER INSERT
Tabla	User
Operativa	Se comprueba si el nuevo usuario del sistema tiene el rol de administrador. En caso de no tenerlo, se incrementa en uno el contador de usuarios no administradores del sistema.

trg_update_user_statistics_on_user_update	
Descripción	Actualiza las estadísticas de usuario tras la actualización de un usuario.
Momento y operación	AFTER UPDATE
Tabla	User
Operativa	Se comprueban los roles antiguos y nuevos del usuario actualizado del sistema: • Si no era administrador y pasa a ser administrador, se decrementa en uno el número de usuarios no administradores del sistema. • Si era administrador y pasa a no serlo, se incrementa en uno el número de usuarios no administradores del sistema. • Actualiza el promedio de transacciones por usuario a través del procedimiento Update_Average_Transactions_Per_User.

trg_update_user_statistics_on_user_delete	
Descripción	Actualiza las estadísticas de usuario tras el borrado de un usuario.
Momento y operación	AFTER DELETE
Tabla	User
Operativa	Se comprueba si el usuario borrado tenía el rol de

	administrador. En caso de no tenerlo, se decrementa en uno el contador de usuarios no administradores del sistema. Por otra parte, actualiza el promedio de transacciones por usuario a través del procedimiento Update_Average_Transactions_Per_User.

trg_update_transaction_statistics_on_transaction_insert	
Descripción	Actualiza las estadísticas de transacciones tras la inserción de una transacción.
Momento y operación	AFTER INSERT
Tabla	Transaction
Operativa	• Se incrementa el contador de transacciones del sistema. • Actualiza el promedio de transacciones por usuario a través del procedimiento Update_Average_Transactions_Per_User. • Actualiza el número de transacciones para el mes de la transacción llamando al procedimiento Update_Transaction_Statistics. • Actualiza la variación de transacciones respecto al último mes llamando al procedimiento Update_Transaction_Variation_Last_Month. • Actualiza las estadísticas de ocupación del evento llamando al procedimiento Update_Event_With_Sales_Occupation.

trg_update_transaction_statistics_on_transaction_update	
Descripción	Actualiza las estadísticas de transacciones tras la actualización de una transacción.
Momento y operación	AFTER UPDATE
Tabla	Transaction
Operativa	• Si en la actualización varía la fecha: ○ Se decrementa en uno el número de transacciones en la fecha anterior mediante el procedimiento

	Update_Transaction_Statistics. ○ Se incrementa en uno el número de transacciones en la fecha actual mediante el procedimiento Update_Transaction_Statistics. ○ Se actualiza el porcentaje de variación de las transacciones entre el mes anterior y el actual mediante el procedimiento Update_Transaction_Variation_Last_Month. • Actualiza la ocupación del evento mediante el procedimiento Update_Event_With_Sales_Occupation.

trg_update_transaction_statistics_on_transaction_delete	
Descripción	Actualiza las estadísticas de transacciones tras el borrado de una transacción.
Momento y operación	AFTER DELETE
Tabla	Transaction
Operativa	• Se decrementa en uno el contador de transacciones del sistema. • Actualiza el promedio de transacciones por usuario a través del procedimiento Update_Average_Transactions_Per_User. • Decrementa las transacciones del mes mediante el procedimiento Update_Transaction_Statistics. • Actualiza la variación de transacciones del mes actual respecto al anterior mediante el procedimiento Update_Transaction_Variation_Last_Month. • Actualiza las estadísticas de ocupación de los eventos con ventas usando el procedimiento Update_Event_With_Sales_Occupation.

trg_update_occupation_statistics_on_event_with_sales_update	
Descripción	Actualiza las estadísticas de transacciones tras el borrado de una transacción.
Momento y operación	AFTER UPDATE
Tabla	Event_With_Sales

Operativa	• Se comprueba si la nueva capacidad del evento no es nula. Si no lo es, se actualizan las estadísticas de ocupación del evento mediante el procedimiento Update_Event_With_Sales_Occupation.

4.6.3. Tareas programadas

Debido a la necesidad de tener que acceder a los datos estadísticos en tiempo constante 1, consideramos que no podemos directamente consultar los datos estadísticos en una tabla y mostrar su resultado ordenándolos y limitándolos, ya que cuanto mayor sea la cantidad de datos, mayor será el tiempo de procesamiento necesario. Si bien con el uso de índices podría ser posible reducir drásticamente el tiempo de procesamiento, nunca llegaríamos a tener un tiempo constante 1.

Por tanto, para poder seleccionar estos datos, hemos creado una serie de tablas que únicamente almacenan los resultados ya filtrados, limitados y ordenados, de forma que únicamente tengamos que efectuar una sentencia SELECT para recuperarlos.

Ahora bien, esto implica que cada vez que se produzcan modificaciones, para tener estos datos actualizados, tendremos que borrar todos los datos de dicha tabla y volverlos a generar. Esto puede no ser la manera óptima de hacerlo ya que, en caso de que se produzcan muchas operaciones que tengan efecto sobre los datos estadísticos podrían llevar a una saturación del sistema.

Para poder evitar estas situaciones, hemos decidido incorporar pg_cron [24], una extensión de PostgreSQL que nos permite programar tareas de forma recurrente. Para ello, editaremos el fichero Dockerfile e introducimos un comando apt-get que descarga la versión de pg_cron correspondiente a nuestra versión de PostgreSQL:

```
FROM postgres:14

# Install pg_cron
RUN apt-get update && apt-get -y install postgresql-14-cron

# Create different locations for tablespaces
RUN mkdir -p /var/lib/pg_tablespaces/operational_tablespace
RUN mkdir -p /var/lib/pg_tablespaces/warehouse_tablespace

RUN chown -R postgres:postgres /var/lib/pg_tablespaces
```

Figura 7: Contenido del fichero Dockerfile con pg_cron

Tras ello, para el correcto funcionamiento de la extensión, necesitaremos añadir un par de configuraciones al fichero de configuración de PostgreSQL:

- **shared_preloaded_libraries**: es un parámetro de configuración de PostgreSQL que nos indica qué librerías deben ser usadas al iniciarlo. En este caso, indicamos la extensión pg_cron.
- **cron.database_name**: especificaremos el nombre de la base de datos en la que se efectuarán las planificaciones de pg_cron. En este caso, es el nombre de nuestra base de datos (event_database).

Esta configuración aplicada requiere de un reinicio de PostgreSQL para poder aplicarse. Debido a que en nuestro entorno ejecutamos los ficheros de inicialización nada más ponerse en marcha este entorno, no podemos efectuar estas operaciones, por lo que hemos optado por modificar el fichero postgresql.conf.sample para que la configuración se añada al generar el fichero postgresql.conf:

```
12  # Add pg_cron data to postgresql config
13  RUN echo "shared_preload_libraries='pg_cron'" >> /usr/share/postgresql/postgresql.conf.sample
14  RUN echo "cron.database_name='event_database'" >> /usr/share/postgresql/postgresql.conf.sample
```

Figura 8: Sentencias de añadido de los parámetros de pg_cron

Por último, deberemos activar en PostgreSQL la extensión de pg_cron:

```
1  \c event_database;
2
3  CREATE EXTENSION IF NOT EXISTS pg_cron;
4
```

Figura 9: Activación de la extensión pg_cron en PosgreSQL

Para el correcto funcionamiento del cron, crearemos una serie de procedimientos que serán los que programaremos para ejecutar cuando sea necesario:

Update_Top_Commented_Events	
Descripción	Actualiza la tabla de eventos más comentados.
Parámetros de entrada	No dispone.
Operativa	• Vacía la tabla de eventos más comentados. • Inserta en la tabla los 10 eventos más comentados.

Update_Top_Valued_Events	
Descripción	Actualiza la tabla de eventos más valorados.
Parámetros de entrada	No dispone.
Operativa	• Vacía la tabla de eventos más valorados. • Inserta en la tabla los 10 eventos más valorados.

Update_Top_Sold_Events	
Descripción	Actualiza la tabla de eventos más vendidos.
Parámetros de entrada	No dispone.
Operativa	• Vacía la tabla de eventos más vendidos. • Inserta en la tabla los 10 eventos más vendidos.

Update_Top_Locations_With_Events	
Descripción	Actualiza la tabla de recintos con más eventos.
Parámetros de entrada	No dispone.
Operativa	• Vacía la tabla de recintos con más eventos. • Inserta en la tabla los 20 recintos con más eventos.

Update_Top_Cities_With_Events	
Descripción	Actualiza la tabla de ciudades con más eventos.
Parámetros de entrada	No dispone.
Operativa	• Vacía la tabla de ciudades con más eventos. • Inserta en la tabla las 10 ciudades con más eventos.

Update_Top_Favorite_Events	
Descripción	Actualiza la tabla de eventos más marcados como favoritos.
Parámetros de entrada	No dispone.
Operativa	• Vacía la tabla de eventos más marcados

	como favoritos. • Inserta en la tabla los 20 eventos más marcados como favoritos.

Una vez hecho esto, programaremos la ejecución de los eventos. Para ello, se usa la sintaxis básica de CRON, que tiene la siguiente estructura:
* * * * *

En la que:

- El primer valor se corresponde con el minuto.
- El segundo valor se corresponde con la hora.
- El tercer valor se corresponde con el día del mes.
- El cuarto valor se corresponde con el mes.
- El quinto valor se corresponde con el día de la semana.

Cuando indicamos el valor con un asterisco, estamos indicando que queremos que se ejecute para todos los valores.

La frecuencia de ejecución de las tareas programadas dependerá de diversos factores, como pueden ser las necesidades de negocio que tenga nuestro proyecto (la frecuencia con la que consultamos y necesitamos esos datos actualizados) o técnicas (programar los eventos estadísticos que más carga puedan trasladar a la base de datos en momentos en los que la carga del servidor sea más baja para evitar saturaciones y caídas del servicio).

En este caso, hemos decidido programar todos los eventos a las 12 de la noche, separados entre ellos en un minuto para evitar su ejecución simultánea:

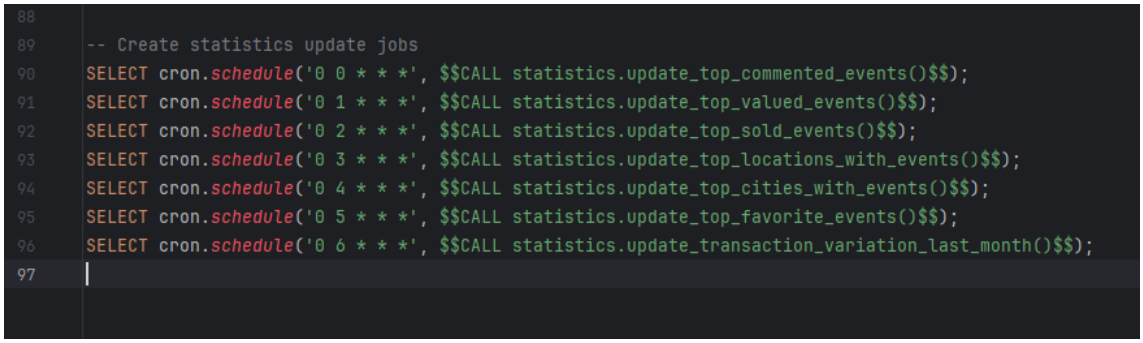


Figura 10: Programación de los eventos en pg_cron

Como podemos apreciar, en el primer atributo de la función “Schedule” indicamos cuál es la frecuencia de actualización y en el segundo la acción a realizar.

A continuación, podemos comprobar que las tareas han sido programadas de forma correcta:

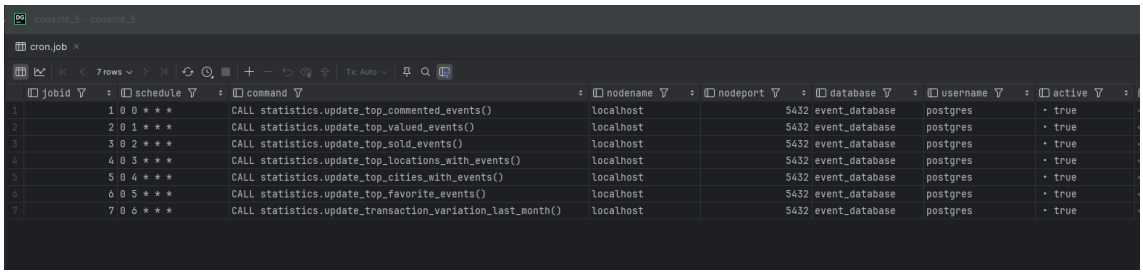


Figura 11: Listado de eventos programados

4.6.4. Vistas

Podemos definir las vistas como relaciones virtuales representadas por su nombre y definición que no existen físicamente en la base de datos.

En este caso, las vistas nos servirán para simplificar algunas de las consultas que el usuario tiene que hacer para recuperar los datos del repositorio estadístico.

Las vistas que hemos definido son las siguientes:

Vista	Dato que recupera
total_non_admin_users	Número total de usuarios no administradores del sistema.
average_event_price	Precio medio de los eventos.
average_transactions_per_user	Promedio de transacciones por usuario.
payed_events_percentage	Porcentaje de eventos de pago.
transactions_variation	Variación de transacciones del mes actual respecto al anterior.
average_occupation	Ocupación media de los eventos.
full_events_percentage	Porcentaje de eventos llenos.

4.6.5. Consultas

A continuación, se detallan las diferentes consultas que se deben efectuar para recuperar los datos estadísticos definidos en los requisitos:

Dato estadístico	Consulta
Listado de los 10 eventos con más comentarios	SELECT * FROM statistics.top_commented_events;

Listado de los 10 eventos con mejor valoración	SELECT * FROM statistics.top_valued_events;
Listado de los 10 eventos con más ventas	SELECT * FROM statistics.top_sold_events;
Listado de los 20 recintos con más eventos	SELECT * FROM statistics.top_event_locations;
Listado de los 20 eventos más marcados como favoritos por los usuarios.	SELECT * FROM statistics.top_favorite_events;
Listado de las 10 ciudades con mayor número de eventos	SELECT * FROM statistics.top_event_cities;
Precio medio de los eventos de pago	SELECT * FROM statistics.average_event_price;
Número total de usuarios no administradores registrados en el sistema	SELECT * FROM statistics.total_non_admin_users;
Promedio del número de transacciones por usuario del sistema	SELECT * FROM statistics.average_transactions_per_user;
Porcentaje de variación del número de transacciones durante el último mes respecto al anterior	SELECT * FROM statistics.transactions_variation;
Porcentaje medio de ocupación de los eventos con venta de entradas.	SELECT * FROM statistics.average_occupation;
Porcentaje de eventos con entradas agotadas.	SELECT * FROM statistics.full_events_percentage;
Porcentaje de eventos con el sistema de pago activo.	SELECT * FROM statistics.payed_events_percentage;

4.7. Pruebas

4.7.1. Framework para pruebas: pgTAP

A la hora de efectuar nuestras pruebas en la base de datos, hemos decidido introducir un software que nos ayude a efectuar una serie de pruebas unitarias y automatizadas: pgTAP.

De esta manera, usando aserciones, podremos comprobar con facilidad que los desarrollos que hemos realizado cumplen con los criterios de validación que hemos especificado y que, por tanto, los requisitos que hemos definido se cumplen.

Por otra parte, nos servirá para hacer pruebas de regresión para comprobar que futuros cambios que efectuemos en la base de datos no rompen funcionalidad previamente desarrollada y, en caso de hacerlo, poder tomar las medidas oportunas para resolver dichos fallos.

Para instalar pgTAP, tenemos que volver a editar el fichero Dockerfile e introducir un comando apt-get que descargue la versión de pgTAP correspondiente a la de nuestra versión de PostgreSQL:

```
FROM postgres:14

# Install pg_cron and pgTap
RUN apt-get update && apt-get -y install postgresql-14-cron postgresql-14-pgtap

# Create different locations for tablespaces
RUN mkdir -p /var/lib/pg_tablespaces/operational_tablespace
RUN mkdir -p /var/lib/pg_tablespaces/warehouse_tablespace

RUN chown -R postgres:postgres /var/lib/pg_tablespaces

# Add pg_cron data to postgresql config
RUN echo "shared_preload_libraries='pg_cron'" >> /usr/share/postgresql/postgresql.conf.sample
RUN echo "cron.database_name='event_database'" >> /usr/share/postgresql/postgresql.conf.sample

```

Figura 12: Contenido del fichero Dockerfile con pgTAP

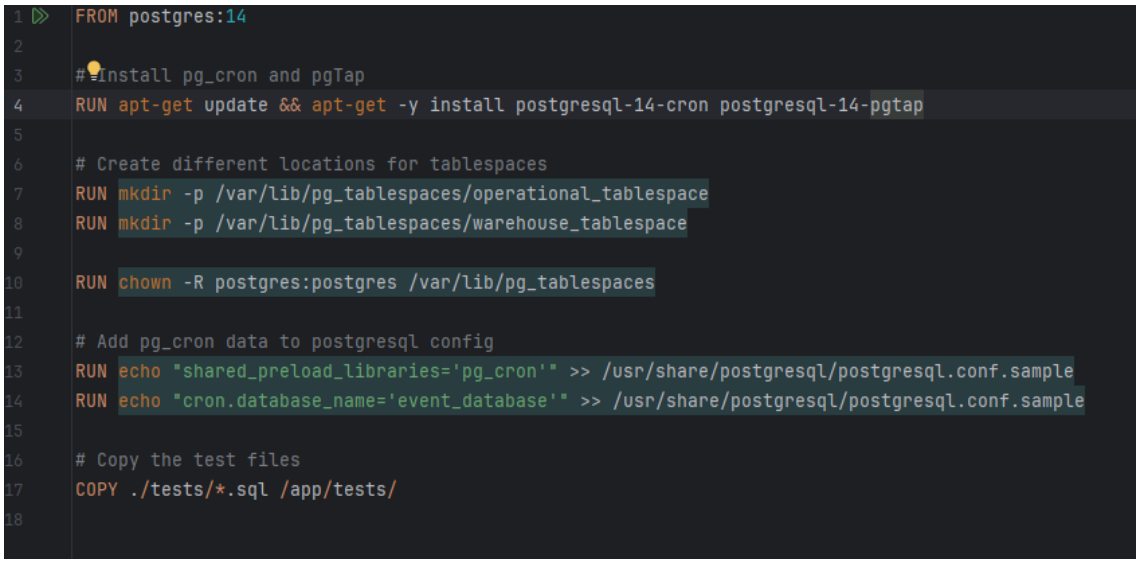
Una vez creada la base de datos, se activará en PostgreSQL la extensión de pgTAP:

```
\c event_database;

CREATE EXTENSION IF NOT EXISTS pgtap;
CREATE EXTENSION IF NOT EXISTS pg_cron;

```

Figura 13: Activación de la extensión pgTAP en PostgreSQL

Todas las pruebas las programaremos en ficheros SQL que ubicaremos en la carpeta raíz “tests”. Ejecutaremos las pruebas automatizadas dentro del contenedor usando la herramienta “pg_prove”, que nos permitirá ejecutar uno a uno todos los ficheros de pruebas que definamos y nos indicará cuándo se produce un error y en qué prueba.

Para añadir los ficheros SQL al contenedor modificamos el fichero Dockerfile para introducir la instrucción “COPY” de la siguiente manera:



```
FROM postgres:14

#Install pg_cron and pgTap
RUN apt-get update && apt-get -y install postgresql-14-cron postgresql-14-pgtap

# Create different locations for tablespaces
RUN mkdir -p /var/lib/pg_tablespaces/operational_tablespace
RUN mkdir -p /var/lib/pg_tablespaces/warehouse_tablespace

RUN chown -R postgres:postgres /var/lib/pg_tablespaces

# Add pg_cron data to postgresql config
RUN echo "shared_preload_libraries='pg_cron'" >> /usr/share/postgresql/postgresql.conf.sample
RUN echo "cron.database_name='event_database'" >> /usr/share/postgresql/postgresql.conf.sample

# Copy the test files
COPY ./tests/*.sql /app/tests/

```

Figura 14: Copiado de los ficheros de prueba al contenedor

Para poner en marcha las pruebas, ejecutaremos el comando pg_prove /app/tests/*.sql.

4.7.2. Documento de pruebas

Se han preparado una serie de pruebas unitarias que se pueden ejecutar en el proyecto. El detalle y resultado de las diferentes pruebas viene especificado en el Anexo I.

Tras ejecutar todas las pruebas de forma automática, se comprueba que todos los tests creados se han ejecutado de forma correcta, lo que validaría la implementación probada:

```
/app/tests/47_statistics_jobs.sql ..........
1..12
ok 1 - Top commented events updated
ok 2 - Top commented events updated in correct order
ok 3 - Top valued events updated
ok 4 - Top valued events updated in correct order
ok 5 - Top sold events updated
ok 6 - Top sold events updated in correct order
ok 7 - Top event locations updated
ok 8 - Top event locations updated in correct order
ok 9 - Top event cities updated
ok 10 - Top event cities updated in correct order
ok 11 - Top favorite events updated
ok 12 - Top favorite events updated in correct order
ok
All tests successful.
Files=47, Tests=610,  3 wallclock secs ( 0.28 usr  0.06 sys +  1.43 cusr  0.28 csys =  2.05 CPU)
Result: PASS
root@cde459548df5:/#
```

Figura 15: Resultado de la ejecución de las pruebas

5. Conclusiones

El desarrollo de este proyecto nos ha permitido alcanzar el objetivo de crear una base de datos orientada a la gestión de eventos y venta de entradas de manera satisfactoria, aplicando conocimientos y habilidades transversales adquiridas durante toda la duración del Grado en Ingeniería Informática.

Desde el punto de vista de la gestión de proyectos, hemos decidido usar el modelo en cascada ya que consideramos que los requisitos iniciales y las fechas tenían poca flexibilidad al cambio. Si bien esto es cierto, como punto negativo implicaba que teníamos poca tolerancia al cambio y por tanto la planificación debía ser efectuada de manera realista.

En términos generales hemos cumplido todos los objetivos y en su mayoría hemos conseguido entregar todos los puntos en los plazos dados, sin embargo, durante el desarrollo, hemos comprobado que algunas de las estimaciones eran demasiado optimistas y los requisitos tenían implicaciones mucho más profundas que las estimadas, necesitando una inversión mucho mayor de tiempo. Afortunadamente, con una dedicación mayor se han podido paliar los efectos que esto tuvo sobre el proyecto.

El mayor contratiempo surgió durante la entrega de la PEC3, ya que no pudimos completar el desarrollo del repositorio estadístico tal y como estaba previsto inicialmente. Las principal causa, además de los mayores tiempos de desarrollo de los apartados anteriores, ha sido la incorporación un sistema de pruebas unitarias que nos ha permitido probar y corregir las funcionalidades desarrolladas.

Consideramos, por tanto, que, aunque no estuviera indicado de esa forma en la planificación, priorizar la realización de las pruebas y la corrección en el propio momento del desarrollo de los errores detectados ha sido un acierto ya que ha sido una gran ayuda a la hora de desarrollar un software sólido que pudiera cumplir con los requisitos del proyecto. Nos ha permitido validar los requisitos desarrollados y proteger nuestro desarrollo de cambios que pudieran afectar a puntos previamente completados.

En retrospectiva, por tanto, consideramos que el modelo en cascada fue el adecuado dado el contexto del proyecto, pero que el tener una mayor experiencia en la planificación y toma de requisitos del proyecto hubiera permitido una mejor gestión del tiempo y los recursos y un mejor cumplimiento de los objetivos. Sin embargo, en el contexto de otros proyectos en los que las fechas pudieran ser más flexibles o existiera mayor incertidumbre en los requisitos, el uso de metodologías ágiles podría ser un acierto.

Como conclusión, creemos que el trabajo efectuado en este proyecto ha permitido la puesta en práctica de una amplia cantidad de habilidades adquiridas en bases de datos, mejorar habilidades esenciales en la gestión de proyectos y a conocer nuevas tecnologías y sistemas con los que no había trabajado, resultando en una experiencia muy enriquecedora.

6. Trabajo futuro

En el contexto de la realización de un Trabajo Final de Grado, se han asumido simplificaciones del trabajo a realizar que en un futuro tendrían que terminar de desarrollarse. Algunos de estos puntos son:

- Respecto a la venta de entradas, hemos asumido una cifra de aforo completo a partir de la cual se paraliza una venta de entradas. La realidad con la que nos encontramos es que la venta puede ser muy diferente en función del evento y del recinto. Por ejemplo, podemos encontrarnos con eventos que tengan lugar en muchos días diferentes o incluso por franjas horarias específicas en un mismo día, con precios diferentes en función de zonas reservadas, que permitan la selección de asientos específicos…

- En estos momentos, sólo se permite la realización de una transacción por evento por usuario. En un futuro, se podría permitir la realización de más transacciones por evento por usuario, si bien deberían controlarse el número de entradas vendidas por usuario para poder gestionar los datos del evento.

- Se podría almacenar una mayor cantidad de datos del usuario, un número de teléfono o direcciones físicas (lo cual puede ser particularmente útil en caso de necesitar enviar una entrada física o para poder establecer una dirección de facturación). Al tener una mayor cantidad de métodos de contacto, el usuario también podría seleccionar cuál es su método de comunicación favorito.

- Establecer un sistema de códigos de descuento y permitir la trazabilidad de los descuentos, es decir, poder seguir qué usuarios han utilizado descuentos y para qué transacciones, y cuál ha sido el descuento total aplicado. Además, los descuentos podrán tener condiciones específicas de uso.

- Aunque las pruebas se han hecho pensando en cubrir el máximo de casos posibles, se pueden hacer mejoras que permitan cubrir más casos de la mejor forma posible. Esto ayudará al futuro mantenimiento de la aplicación. Por otra parte, además de las pruebas de procedimientos, también se podrían probar las propias estructuras de las tablas creadas.

7. Glosario

- **SGBD**: Sistema Gestor de Bases de Datos. Es un software específico que permite la administración, mantenimiento y uso de bases de datos.
- **Clave primaria**: campo o conjunto de campos en la base de datos que permite identificar cada fila de la tabla de manera única.
- **Clave alternativa o candidata**: campo o campos que podrían servir como clave primaria pero no han sido seleccionadas como tal.
- **Procedimiento:** funciones o métodos que se pueden usar de forma individual y realizan una tarea específica.
- **Disparador:** procedimiento almacenado que se ejecuta automáticamente cuando tiene lugar un evento específico en la base de datos
- **Prueba unitaria:** una prueba en la que se verifica el funcionamiento de una unidad individual de código para comprobar su correcto funcionamiento de forma aislada. Nos permite garantizar, además de su correcto funcionamiento, el cumplimiento de los requisitos y nos protege frente a cambios de código, mejorando la mantenibilidad de éste.
- **UML:** Unified Modeling Language, se trata de un lenguaje estandarizado de modelado usado en el ámbito del software, diseñado para crear representaciones visuales de sistemas complejos.
- **Base de datos relacional:** se trata de un tipo de bases de datos en la que la información se almacena en tablas que pueden estar interconectadas a través de relaciones preestablecidas.
- **Framework:** conjunto herramientas y librerías de trabajo que aportan al desarrollador un entorno estandarizado para facilitar el desarrollo y mantenimiento de aplicaciones.
- **Repositorio estadístico:** base de datos en la que se almacenan diferentes datos estadísticos recopilados por la aplicación.
- **Cron**: programa de los sistemas operativos tipo Unix que permite programar la ejecución de comandos en momentos específicos y de forma regular.
- **Modelo en cascada:** procedimiento de gestión de proyectos en el que se adopta un enfoque lineal con fases secuenciales y dependientes entre si. Su estructura es rígida y poco tolerante al cambio al deber completarse cada fase para pasar a la siguiente.

8. Bibliografía

1. Hosteltur (2023) *La actividad de las ferias y congresos alcanza niveles de 2019*. Disponible en: (Consultado: 7 de marzo de 2024).
2. Lucidchart *Tutorial de estructura y diseño de bases de datos*. Disponible en: https://www.lucidchart.com/pages/es/tutorial-de-estructura-y-diseno-de-bases-de-datos (Consultado: 10 de marzo de 2024).
3. Asana (2024) *Las 12 metodologías más populares para la gestión de proyectos*. Disponible en: https://asana.com/es/resources/project-management-methodologies (Consultado: 10 de marzo de 2024).
4. GanttPro (2023) *Modelo de cascada (Waterfall): qué es y cuándo conviene usarlo*. Disponible en: https://blog.ganttpro.com/es/metodologia-de-cascada/ (Consultado: 10 de marzo de 2024).
5. Lucidchart. *Análisis del modelo de cascada para la gestión de tus proyectos*. Disponible en: https://www.lucidchart.com/blog/es/metodologia-gestion-proyectos-cascada (Consultado: 10 de marzo de 2024).
6. Pradel Miquel, J. y Raya Martos, J. *Introducción a la ingeniería de requisitos*. [Recurso de aprendizaje textual]. Primera edición. Barcelona. Fundació per la Universitat Oberta de Catalunya (FUOC) (Consultado: 13 de marzo de 2024).
7. Pradel Miquel, J. y Raya Martos, J. *Análisis UML*. [Recurso de aprendizaje textual]. Primera edición. Barcelona. Fundació per la Universitat Oberta de Catalunya (FUOC) (Consultado: 31 de marzo de 2024).
8. Casas Roma, J. y Cuartero Olivera, J. (2020). *Diseño conceptual de bases de datos*. [Recurso de aprendizaje textual]. Quinta edición. Barcelona. Funció per la Universitat Oberta de Catalunya (FUOC) (Consultado: 31 de marzo de 2024).
9. Burgués Illa, X. y Cuartero Olivera, J. (2020). *Diseño lógico de bases de datos*. [Recurso de aprendizaje textual]. Quinta edición. Barcelona. Fundació per la Universitat Oberta de Catalunya (FUOC) (Consultado: 6 de abril de 2024).
10. Cabré y Segarra, B., Casas Roma, J., Costal Costa, D., Plana Vallvé, I., Rius Gavididia, A. y Segret i Sala, R. (2020). *Diseño lógico de bases de datos*. [Recurso de aprendizaje textual]. Séptima edición. Barcelona. Fundació per la Universitat Oberta de Catalunya (FUOC) (Consultado: 9 de abril de 2024).
11. Mateos Bartolomé, A., Esteban Grifoll, J. y Panadero Martínez, J. (2020). *Administración de servicios web*. [Recurso de aprendizaje textual]. Primera edición. Barcelona. Fundació per la Universitat Oberta de Catalunya (FUOC) (Consultado: 6 de mayo de 2024).
12. AWS. *Contenedores de Docker | ¿Qué es Docker?*. Disponible en: https://aws.amazon.com/es/docker/ (Consultado: 6 de mayo de 2024).
13. Techtarget. *What is Docker and how does it work?*. Disponible en: https://www.techtarget.com/searchitoperations/definition/Docker (Consultado: 8 de mayo de 2024).

14. KeepCoding (2023): *¿Qué es Docker Compose?*. Disponible en: https://keepcoding.io/blog/que-es-docker-compose/ (Consultado: 8 de mayo de 2024).
15. Docker. *What is an image?*. Disponible en: https://docs.docker.com/guides/docker-concepts/the-basics/what-is-an-image/ (Consultado: 8 de mayo de 2024).
16. Docker Hub. *Postgres – Official Image*. Disponible en: https://hub.docker.com/_/postgres (Consultado: 8 de mayo de 2024).
17. Cabré i Segarra, B., Casas Roma, J., Costal Costa, D., Juanola Juanola, P., Plana Vallvé, I., Rius Gavidia, À y Segret i Sala, R. (2020). *Diseño físico de bases de datos*. [Recurso de aprendizaje textual]. Séptima edición. Barcelona. Fundació per la Universitat Oberta de Catalunya (FUOC) (Consultado: 8 de mayo de 2024).
18. KeepCoding. *¿Qué son las ramas en git y para qué sirven?*. Disponible en: https://keepcoding.io/blog/que-son-las-ramas-en-git/ (Consultado: 9 de mayo de 2024).
19. GitHub. *Acerca de las ramas*. Disponible en: https://docs.github.com/es/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches (Consultado: 9 de mayo de 2024).
20. Pérez Braña, J., Jové Canela, A., Ortego Carazo, S. y Plana Vallé, I. (2022). *Procesamiento de consultas y vistas*. [Recurso de aprendizaje textual]. Sexta edición. Barcelona. Fundació per la Universitat Oberta de Catalunya (FUOC) (Consultado: 15 de mayo de 2024).
21. Martín Escofet, C. *El lenguaje SQL I* [Recurso de aprendizaje textual]. Barcelona. Fundació per la Universitat Oberta de Catalunya (FUOC) (Consultado: 15 de mayo de 2024).
22. Martín Escofet, C. *El lenguaje SQL II* [Recurso de aprendizaje textual]. Barcelona. Fundació per la Universitat Oberta de Catalunya (FUOC) (Consultado: 15 de mayo de 2024).
23. pgTAP. pgTAP: *Unit testing for PostgreSQL*. Disponible en: https://pgtap.org/ (Consultado: 17 de mayo de 2024).
24. CitusData. *pg-cron*. Disponible en: https://github.com/citusdata/pg_cron (Consultado: 19 de mayo de 2024).
25. Sabbane. *How to customize the configuration file of the official PostgreSQL Docker Image?*. Disponible en: https://stackoverflow.com/questions/30848670/how-to-customize-the-configuration-file-of-the-official-postgresql-docker-image (Consultado: 31 de mayo de 2024).

9. Anexos

I. Documento de pruebas
II. Diagrama de Gantt completo del proyecto.
III. Código del proyecto.
IV. Presentación.
V. Informe de autoevaluación.