

TP GDD 2C2018

Estrategia

PalcoNet



Peaky Blinders

20 de Diciembre de 2018

Índice

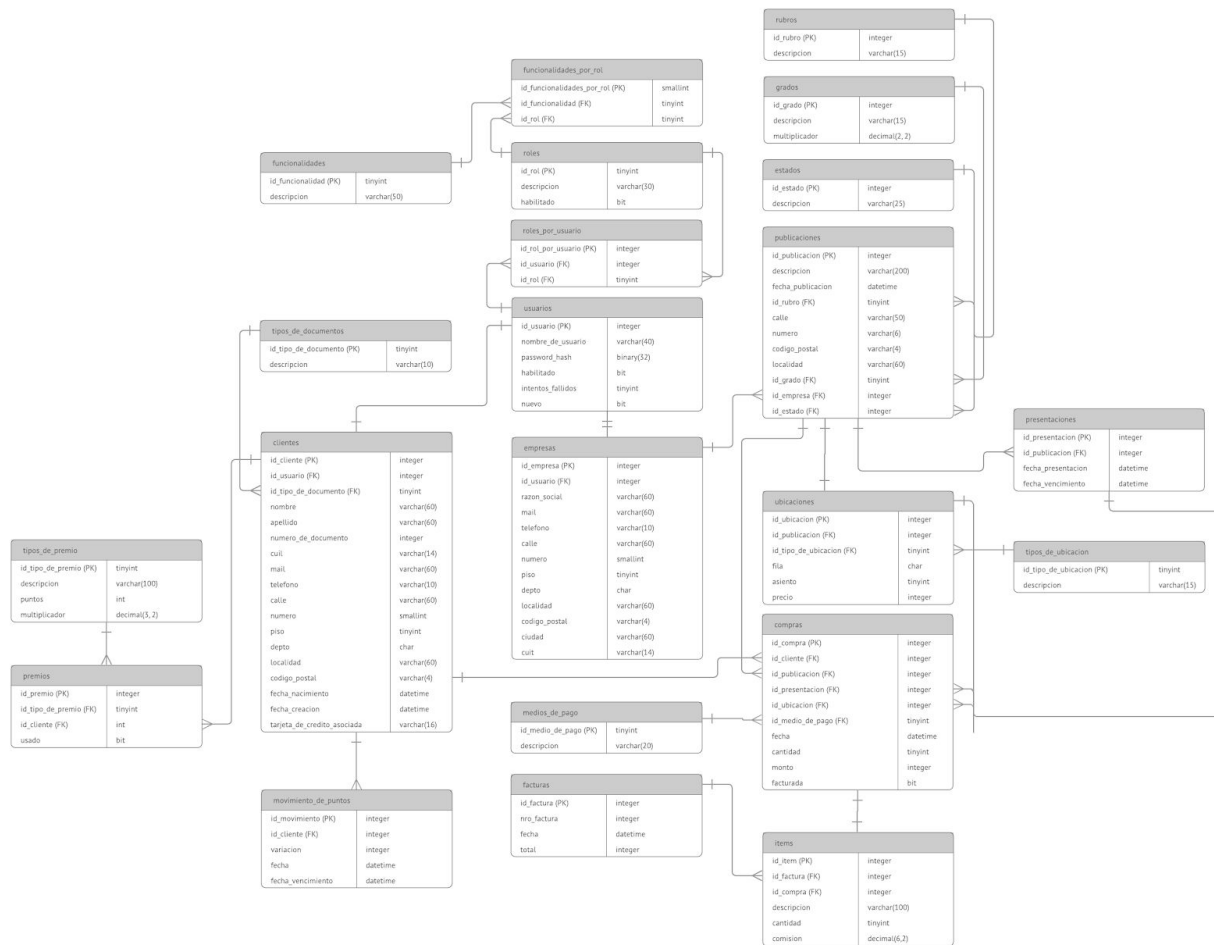
- Estructuras de datos (pág 2)
 - Diagrama Entidad Relación (pág 3)
 - Publicaciones (pág 4)
 - Rubros, Grados y Estados (pág 5)
 - Presentaciones (pág 6)
 - Empresas (pág 7)
 - Usuarios (pág 8)
 - Roles, Roles por usuario, Funcionalidades, Funcionalidades por rol (pág 9)
 - Clientes (pág 11)
 - Tipos de documento y Movimiento de puntos (pág 12)
 - Premios y Tipos de premios (pág 13)
 - Medios de pago y Facturas (pág 14)
 - Ítems (pág 15)
 - Compras (pág 16)
 - Ubicaciones y Tipos de ubicación (pág 17)
- Procedures (pág 18)
- Migración (pág 22)

Estructuras de datos

Luego de leer el enunciado y de ver los datos existentes en la tabla Maestra llegamos al siguiente modelado de datos que nos permitió no perder datos anteriores y manejar las nuevas funcionalidades del sistema.

- Para todas las tablas se decidió utilizar claves subrogadas nombradas de la siguiente manera: "id_{nombre_de_entidad}".
- Para todas las claves foráneas se decidió utilizar el mismo nombre que tenía esa clave en la tabla a la cual se hace referencia.

Diagrama Entidad Relación



Sugerencia: El DER se puede observar mejor en la imagen “der.png” que se encuentra en este proyecto.

Publicaciones

Las publicaciones constan de una gran cantidad de claves foráneas porque contienen mucha información que se repite entre muchas de ellas. La tabla está estructurada de la siguiente manera:

Tiene una columna **id_publicacion** como clave primaria y cuatro claves foráneas:

1. **id_estado** establece la relación con la tabla “estados” con el id del estado en el cual se encuentra la publicación en ese momento.
2. **id_grado** establece la relación con la tabla “grados” con el id del grado (%) que se utilizara.
3. **id_empresa** establece la relación con la tabla “empresas” con el id de la empresa que hizo la publicación.
4. **id_rubro** establece la relación con la tabla “rubros” con el id del rubro al cual pertenece el espectáculo presentado en esta publicación.

Además tiene otras columnas que completan la información sobre el espectáculo publicado; su descripción, lugar donde se hará. También guarda la fecha de la publicación.

publicaciones	
id_publicacion (PK)	integer
dsecripcion	varchar(200)
fecha_publicacion	datetime
id_rubro (FK)	tinyint
calle	varchar(50)
numero	varchar(6)
codigo_postal	varchar(4)
localidad	varchar(60)
id_grado (FK)	tinyint
id_empresa (FK)	integer
id_estado (FK)	integer

Rubros

La tabla de rubros consta de una columna **id_rubro** y una columna **descripcion** que se utiliza para representar si la obra es de 'Comedia', 'Drama', 'Musical' u cualquier otro rubro posible.

En el caso de la tabla maestra, ninguna publicación tenía rubro, por lo que se decidió para favorecer la integridad y consistencia de la base, asignarles el rubro "Otros".

Grados

La tabla de grados tiene una columna **id_grado** y una columna **multiplicador** que es un número decimal menor a 1 y con precisión de dos dígitos que representa el valor por el cual hay que multiplicar el monto de una compra para obtener la comisión que deberá facturar el sistema. Por ejemplo, para todas las publicaciones que se encontraban en la tabla Maestra se decidió tomar un grado con multiplicador igual a '0.10'. Además se agrega una columna **descripcion** para mostrar al momento de elegir un grado.

Estados

La tabla de estados es bastante simple ya que tiene la columna **id_estado** y una columna **descripcion** donde se podrán encontrar valores como 'Borrador', 'Publicada', 'Finalizada'.

rubros	
id_rubro (PK)	integer
descripcion	varchar(15)

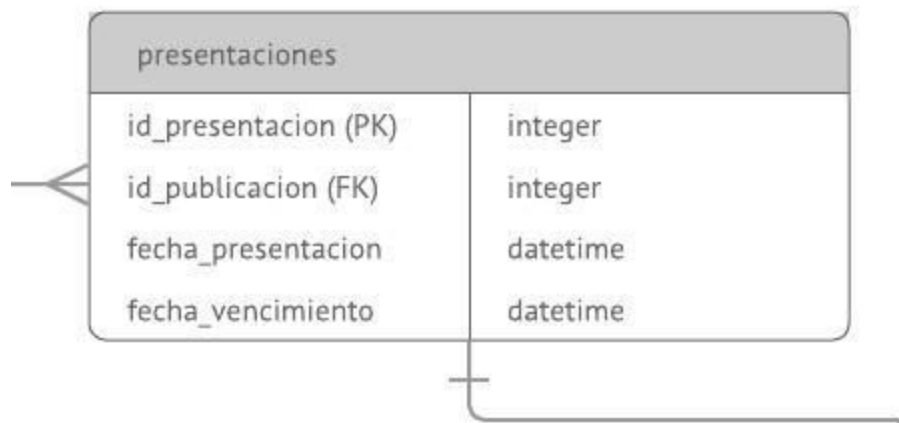
grados	
id_grado (PK)	integer
descripcion	varchar(15)
multiplicador	decimal(2, 2)

estados	
id_estado (PK)	integer
descripcion	varchar(25)

Presentaciones

La tabla de presentaciones existe para poder cumplir con la funcionalidad de tener X cantidad de horarios-fecha para cada publicación, sin tener que repetir los datos de la publicación y así simplificar la tarea de editarla.

Tiene una columna **id_presentacion** que actúa de clave primaria, una columna **id_publicacion** que referencia a la tabla de publicaciones bajo el id de la publicación que da origen a ese evento en el cual se va a presentar la obra/espectáculo. Tiene una columna **fecha_presentacion** que corresponde al momento en el cual va a comenzar el evento. Por último tiene una **fecha_vencimiento** que refiere a la fecha a partir de la cual las ubicaciones dejan de estar disponibles para la compra, consideramos (viendo los datos de la tabla Maestra) que esto se debe hacer 7 (siete) días antes que la **fecha_presentacion**.



Empresas

La tabla empresas contiene una gran cantidad de información en cada fila. Para comenzar define una columna **id_empresa** (PK), una columna **id_usuario** que referencia a la tabla de usuarios. Y muchas columnas más.

Ni la **localidad**, ni la **ciudad** aparecen como datos en la tabla Maestra, con lo cual se decidió dejar estos campos en NULL para las filas migradas.

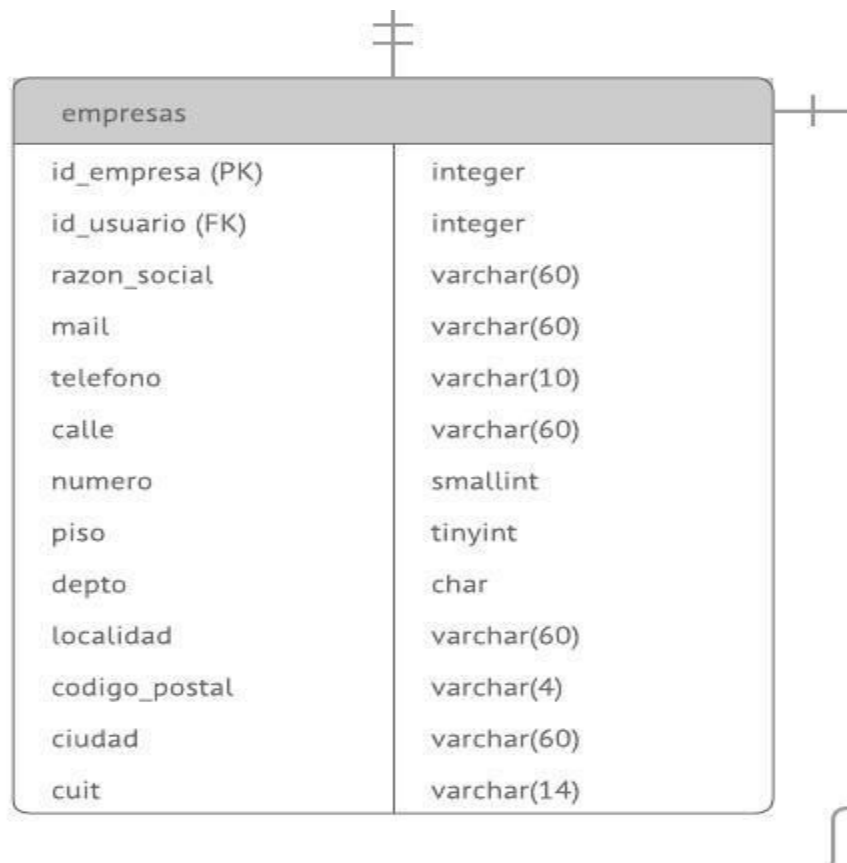
A las empresas migradas se les generaron sus usuarios correspondientes:

nombre de usuario: empresa + n° de razón social

El n° de razón social garantiza que no haya repetición posible en los casos existentes. Ejemplo:

Razon Social N°: 10 => empresa10

contraseña: el CUIT (sin guiones) => 879067575277



empresas	
id_empresa (PK)	integer
id_usuario (FK)	integer
razon_social	varchar(60)
mail	varchar(60)
telefono	varchar(10)
calle	varchar(60)
numero	smallint
piso	tinyint
depto	char
localidad	varchar(60)
codigo_postal	varchar(4)
ciudad	varchar(60)
cuit	varchar(14)

Usuarios

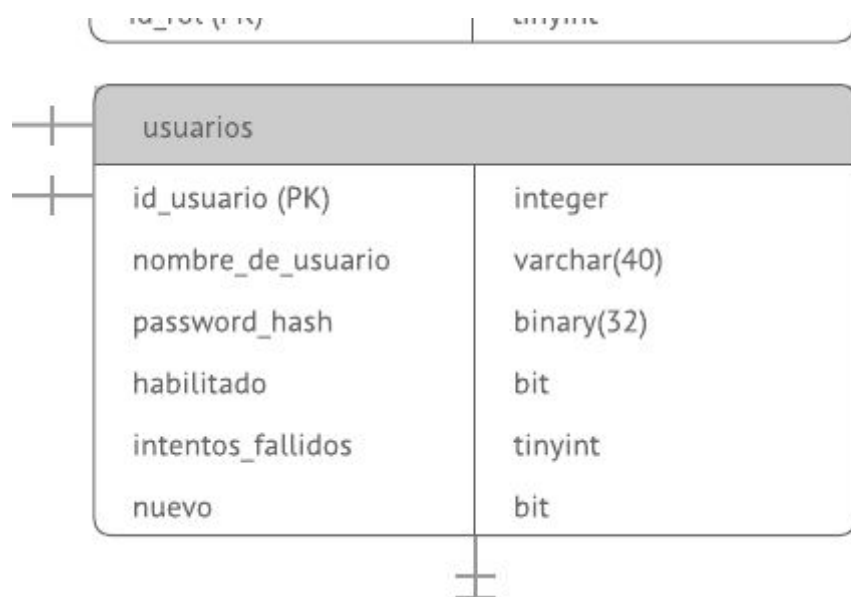
Cada registro de la tabla usuarios representa a un usuario de nuestra aplicación, ya sea un cliente o una empresa. Tiene un `id_usuario`, un `nombre_de_usuario` y un `password_hash`. Estos últimos dos atributos son utilizados para manejar la autenticación de cada uno de ellos. Además de estos campos define `habilitado` y `intentos_fallidos` para poder implementar la funcionalidad bajo la cual se inhabilitan usuarios que hayan tenido fallos consecutivos en el intento de ingreso a la aplicación. Además, el campo `nuevo` permite conocer cuando un usuario tiene que modificar su contraseña al primer login. El campo `habilitado` no solo sirve para persistir si un usuario está habilitado o no según los fallos en el login, sino que también puede deshabilitarse por decisión de un usuario con rol Administrador. En cualquier caso en que el usuario deshabilitado quiera volver a habilitarse, deberá ser por medio del Administrador.

En la migración se genera el usuario con Rol administrador, cuyos datos son:

nombre de usuario: admin

contraseña: admin

(Una vez logueado tendrá la posibilidad de modificar su contraseña).



Roles

La tabla de roles define un **id_rol** y una **descripcion**. De esta manera podemos tener distintos roles como lo son 'Administrador', 'Cliente' y 'Empresa'. También tiene un atributo **habilitado** que nos permite comprobar que el estado esté habilitado.

Roles por usuario

Esta tabla permite que un usuario pueda tener más de un rol en el caso de que sea necesario en un sistema como el nuestro.

Inicialmente el único usuario que tiene todos los roles asignados es "admin".

Aclaración importante: a un usuario registrado como Cliente podría asignársele el rol Empresa e intentar Generar una Publicación, o viceversa y querer Comprar, pero este usuario con ambos roles solo tiene registrados los datos obligatorios de uno de los 2 roles, por lo cual se generarían inconsistencias. Para resolver esto, se diseñó que el caso de uso que permite asignar nuevos roles a un usuario pida los nuevos datos obligatorios para poder procesar esa petición. Toda esta validación de datos no fue realizada, ya que el caso de uso mencionado no fue pedido.

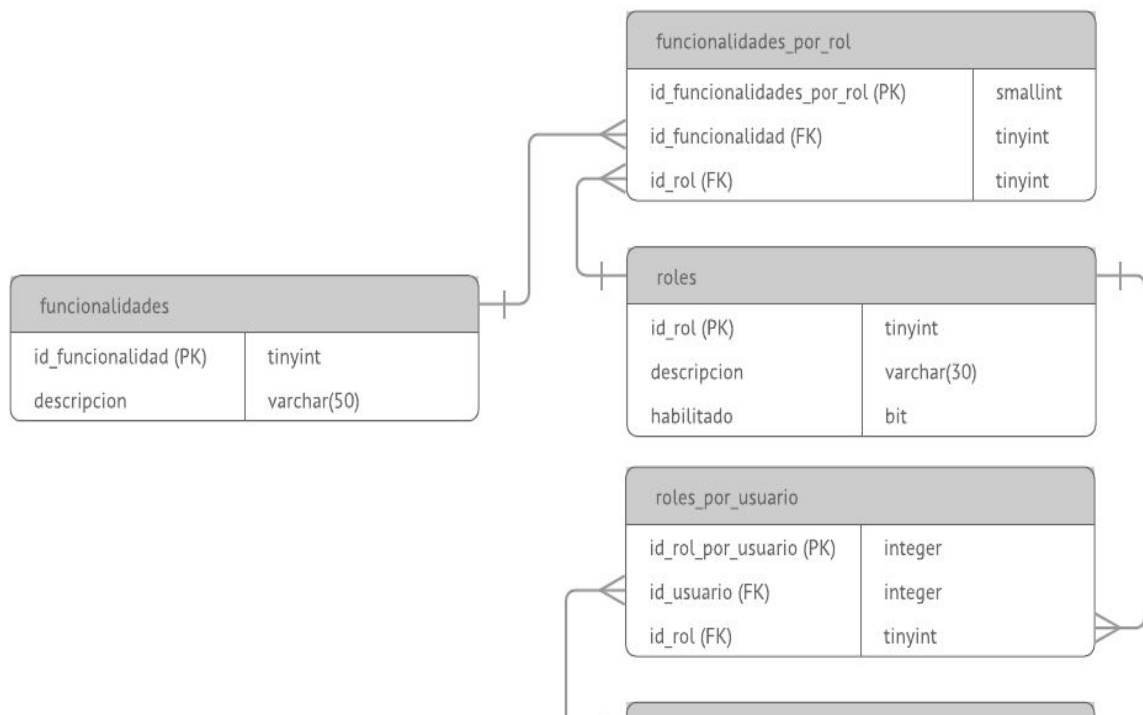
Funcionalidades

Esta tabla permite persistir las distintas funcionalidades del sistema.

Funcionalidades por rol

Esta tabla permite asignar X cantidad de funcionalidades a un rol en particular. En la funcionalidad de ABM Rol se permite adjudicar todas las funcionalidades a cualquier rol existente. Sin embargo, algunas funcionalidades fallarán para determinados roles. Por ejemplo, la funcionalidad Comprar puede ser accedida si y sólo si quien ingresa tiene asignado el rol Cliente, en caso contrario tirará error.

Aclaración: cuando un usuario se loguea, se mostrarán sólo las funcionalidades que tiene asignadas ese rol, ni más ni menos.



Cientes

Los clientes al igual que las empresas tienen un **id_usuario**. También tienen una clave primaria **id_cliente** y una clave foránea **id_tipo_de_documento** que hace referencia a la tabla de tipos de documentos. Además de estas columnas la tabla tiene muchas más que corresponden a toda la información que se requiere sobre un cliente.

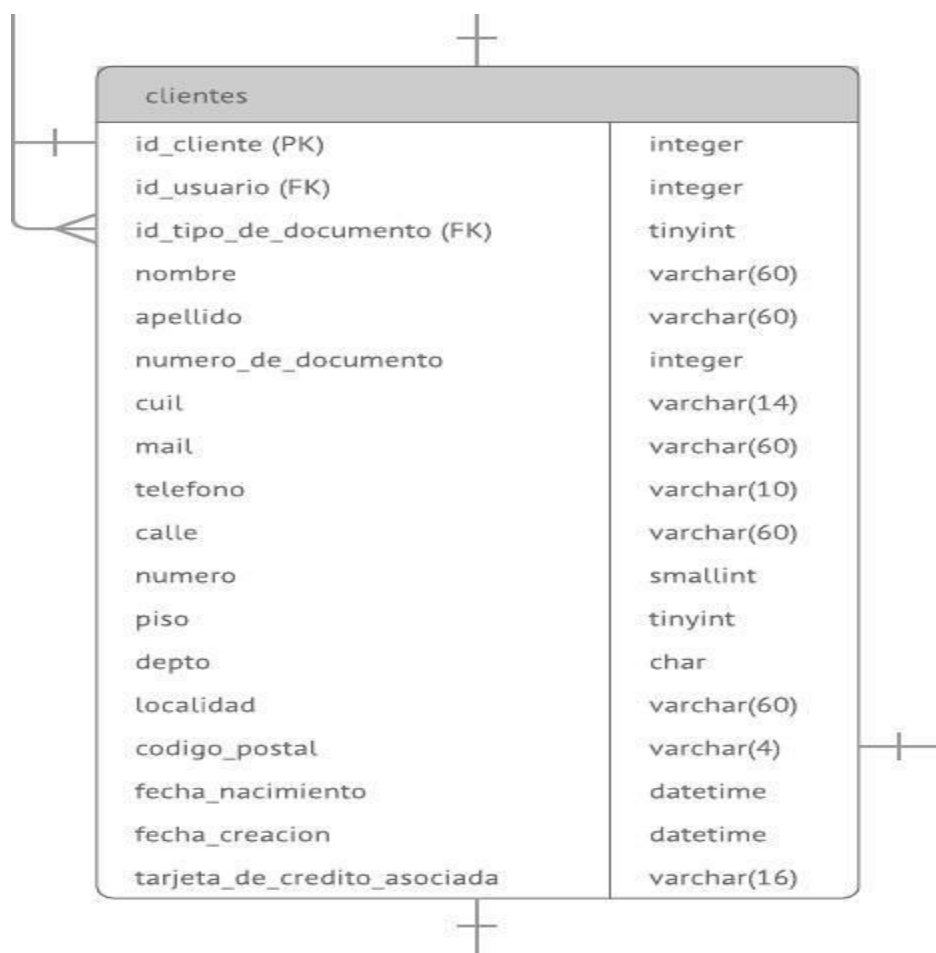
A los clientes migrados se les generaron sus usuarios correspondientes:

nombre de usuario: inicial del nombre + apellido + # + id

El id garantiza que no haya repetición posible. Ejemplo:

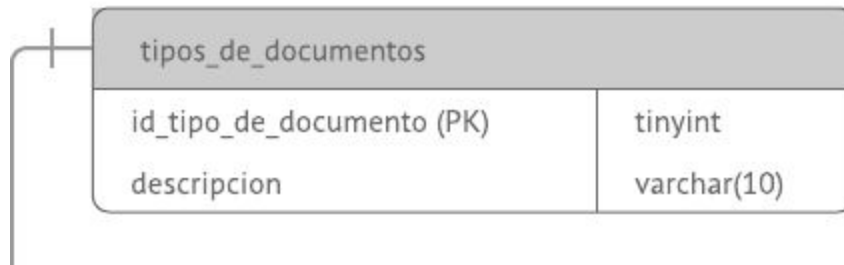
Abdul Vargas => avargas#1

contraseña: DDMMAAAA (fecha de nacimiento) => 15111930



Tipos de documento

Esta tabla persiste los distintos tipos de documento que pueden ser utilizados en nuestro sistema. Ellos son “DNI”, “LC” (Libreta Cívica), “LE” (Libreta de Enrolamiento) y “Pasaporte”.



Movimientos de puntos

Esta tabla permite llevar un registro de los puntos que va ganando o gastando cada cliente. Para eso se tiene una **fecha_vencimiento** que representa hasta cuando serán tenidos en cuenta los mismos y una **variación** de puntos. El cálculo se realiza sumando todas las variaciones según las fechas correspondientes..



Premios y Tipos de premios

Los premios disponibles definimos que sean bonificaciones sobre entradas, se podrá usar un solo premio por cada compra. Existen bonificaciones del 100%, 50% y 25% en una entrada/ubicación.

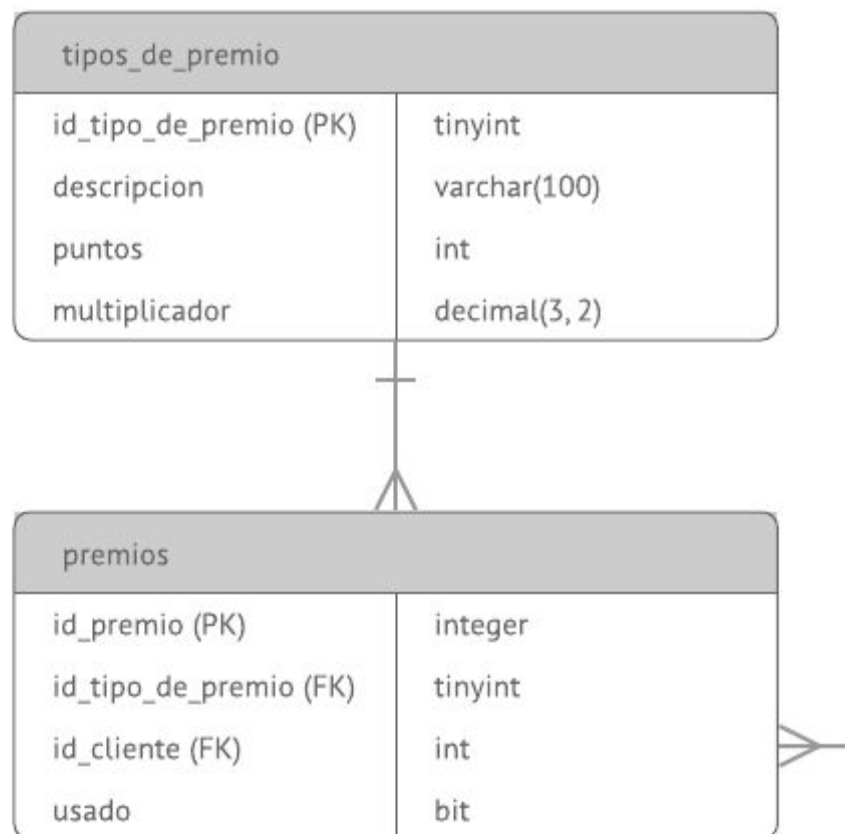
Los puntos se canjean por estos premios:

100% - 1000 puntos

50% - 700 puntos

25% - 400 puntos

El campo **usado** se utiliza para saber si el premio ya fue canjeado.



Medios de pago

En esta tabla están los distintos medios de pago que se utilizan en la plataforma. Contiene a la opción “Efectivo” migrada de la tabla Maestra, y el único medio de pago que acepta momentáneamente la aplicación: “Tarjeta de Crédito”.

medios_de_pago	
id_medio_de_pago (PK)	tinyint
descripcion	varchar(20)

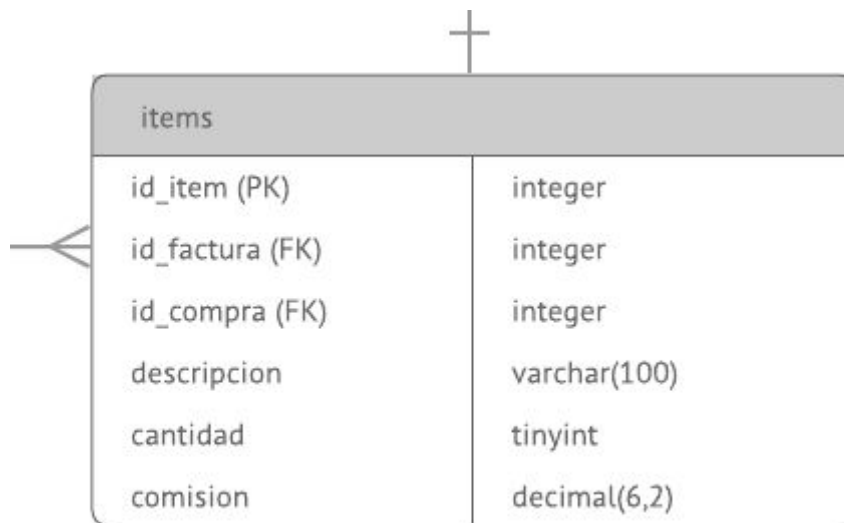
Facturas

La tabla de facturas tiene un **nro_factura**, una **fecha** y un **total**. Cada factura está compuesta por 1 o más items.

facturas	
id_factura (PK)	integer
nro_factura	integer
fecha	datetime
total	integer

Items

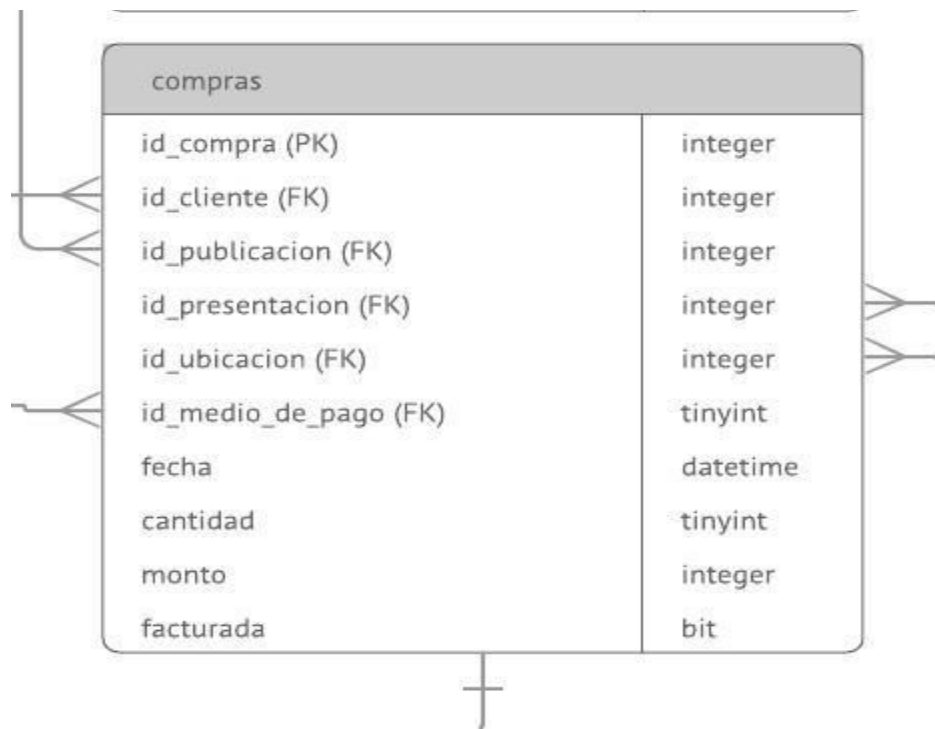
La tabla de ítems está fuertemente relacionada a la tabla de facturas ya que cada ítem tiene un **id_factura** y que cada factura tiene una cierta cantidad de ítems que la componen. Además de ese campo, esta tabla tiene un **id_compra** que referencia a la compra sobre la cual se está facturando ese ítem, una **descripción**, **cantidad** y **comisión**.



Compras

La tabla de compras agrupa claves foráneas de varias tablas por sus campos: **id_cliente**, **id_publicacion**, **id_presentacion**, **id_ubicacion**, **id_medio_de_pago**. También tiene **fecha**, **cantidad** y **monto**. Se usa el id de presentación para saber que esa ubicación se compró para determinada fecha de presentación de un espectáculo.

- Se decidió desnormalizar al poner tanto el id de la presentación como el de la publicación, para facilitar joins tanto en la migración como en consultas del sistema.
- También se decidió replicar el precio de la ubicación comprada por medio del campo monto para evitar que cambios en precios de ubicaciones generen un cambio en compras/facturaciones anteriores.
- Si para la compra se utilizó algún premio canjeado previamente, entonces el descuento será aplicado sobre el mismo **monto**.
- El atributo de **cantidad** se mantuvo para poder migrar la data que venía de la tabla maestra, donde siempre era 1.
- Se utilizó **facturada** para saber si ya fue rendida a la empresa o no.



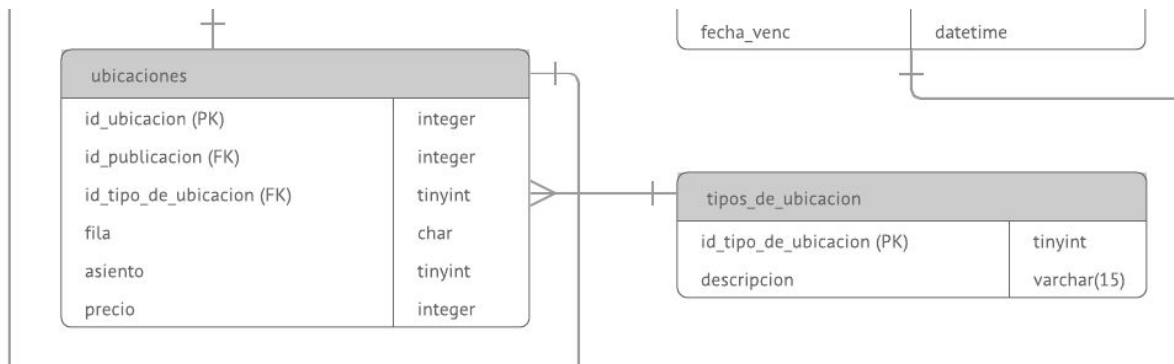
Ubicaciones

La tabla de ubicaciones tiene cada asiento que está a la venta en un evento. Cada ubicación/asiento se corresponde con una publicación. Cada registro es lo más cercano a la representación en sistema de lo que sería un asiento o ubicación física. Tiene un **id_tipo_de_ubicacion**, una **fila**, un **asiento** y un **precio** al cual se vende.

En la tabla Maestra observamos que no todos los asientos del mismo tipo y para el mismo espectáculo costaban lo mismo, con lo cual decidimos poner el precio en cada ubicación. Esto agregó una complejidad y no le encontramos sentido en nuestro sistema por lo cual nos tomamos la libertad de decidir que al momento de crear una publicación el precio quedará fijado por fila o tipo de ubicación y no por cada ubicación en particular. Es decir, se creará una determinada cantidad de “ubicaciones” pero todas tendrán el mismo precio en tanto y en cuanto compartan alguna de las dos características que mencionamos anteriormente (fila, tipo).

Tipos de ubicación

Los tipos de ubicación responden a los que encontramos en la tabla maestra: “Platea Alta”, “Platea Baja”, “Vip”, “Campo”, “Campo Vip”, “PullMan”, “Super PullMan” y “Cabecera”.



Stored Procedures & Functions

Hicimos uso de procedures para separar mucha lógica responsable de insertar o corroborar datos en varias tablas.

autenticar_usuario

Recibe nombre de usuario y contraseña. Retorna un valor que representa si es un nombre de usuario inválido, si la contraseña es incorrecta, si la contraseña es incorrecta y se llegó al 3er intento, si el usuario está inhabilitado, o si está todo ok (y en este caso si el usuario es la primera vez que se loguea o no).

Para esto maneja cada caso posible teniendo en cuenta si la contraseña coincide, cuantos intentos fallidos tiene, si hay que incrementar esos intentos fallidos o limpiarlos.

verificar_contrasenna

Recibe un id_usuario y una contraseña para verificar si al hashearla coincide con la que se encuentra guardada.

actualizar_contrasenna

Recibe un id_usuario y una contraseña y la actualiza sobreescribiendo la anterior.

actualizar_estado_usuario

Recibe un id_usuario y si está habilitado o no, actualizándolo en la base.

Procedures para ABMs

Existen numerosos procedures para generar, actualizar, dar de baja distintas entidades como empresas, clientes, usuarios, publicaciones, presentaciones, roles, funcionalidades por rol, grados, etc.

crear_cliente

Persiste todos los datos de un nuevo cliente incluído su nombre de usuario y contraseña.

modificar_cliente

Actualiza todos los datos del cliente.

baja_cliente

Deshabilita al cliente para acceder al sistema.

cliente_habilitado

Verifica si el cliente está o no habilitado.

registrar_tarjeta

Registra el número de la tarjeta de crédito para el cliente que corresponda.

crear_empresa

Persiste todos los datos de una nueva empresa incluído su nombre de usuario y contraseña.

modificar_empresa

Actualiza todos los datos de la empresa.

baja_empresa

Deshabilita a la empresa para acceder al sistema.

empresa_habilitada

Verifica si la empresa está o no habilitada.

crear_rol

Recibe una descripción y crea un nuevo rol con la misma en caso de que no exista uno igual.

modificar_rol

Modifica la descripción de un rol.

actualizar_rol_habilitado

Actualiza el campo habilitado para un rol particular.

actualizar_funcionalidades_por_rol

Actualiza las funcionalidades que tiene asignadas un rol particular.

generar_publicacion

Genera una nueva publicación con todos sus datos asociados.

modificar_publicacion

Actualiza todos los datos de una publicación en particular con los que son recibidos.

finalizar_publicacion

Pone a una publicación particular en estado de 'Finalizada'.

generar_presentacion

Genera una nueva presentación para una publicación en particular.

generar_ubicacion

Genera una ubicación con sus respectivos datos para cierta publicación. Estos datos son el tipo de ubicación, la fila, el asiento, y el costo.

actualizar_grado

Dada una publicación, actualiza el grado con el que se mostrará.

registrarCompra

Este procedure se encarga de registrar la compra de UNA ubicación (se va a llamar multiples veces si se compra más de una). Corrobora si se eligió usar algún premio, si está habilitado, cuál es multiplicador y por ende el precio a cobrar, registra el cobro y otorga puntos por comprar.

canjear_premio

Maneja el canje de puntos por premios, resta puntos según el premio elegido e inserta en la tabla de premios.



generar_factura

Crea una nueva factura con la fecha de la misma, su número y el monto total.

agregar_item

Dada una factura, agrega ítems que se relacionarán con la misma. Cada ítem tiene su descripción, cantidad, comisión y una compra relacionada.

Migración

Existe un archivo ‘script_creacion_inicial.sql’ que contiene la creación del schema PEAKY_BLINDERS y todas las tablas/procedures que necesitamos. Además realiza la migración de datos de la tabla Maestra.

Para encarar la migración empezamos a observar cuales eran los datos que teníamos en la tabla Maestra y cuáles eran las funcionalidades que utilizaban esos datos. De esa manera fuimos pensando cómo adaptar lo que venía de un diseño/modelado anterior al nuevo modelo que requerían nuestras funcionalidades.

La parte más difícil de adaptar y de terminar de digerir fue la de compras, facturas, ubicaciones e ítems. Llegamos a un modelado que nos permite mantener los datos existentes sin sacrificar el funcionamiento de las funcionalidades que teníamos que implementar.


En la sección de “Estructura de datos” hablamos un poco sobre algunas consideraciones generales en cuanto a, por ejemplo, el precio de las ubicaciones, la idea de publicación y su relación con las múltiples presentaciones. Estos son claros ejemplos de lo comentado en el párrafo anterior.

El precio unitario de las ubicaciones nos permite mantener la información que venía de la tabla Maestra ya que ahí cada ubicación tenía su precio propio. Al mismo tiempo nos permite cumplir con las funcionalidades pedidas mientras hacemos la consideración de que en nuestro sistema todas las ubicaciones que compartan fila o tipo de ubicación para una misma publicación, compartirán el precio; haciendo, así, mucho más simple la carga de ubicaciones y precios para cada publicación.

La idea de tener tanto la publicación como sus múltiples presentaciones nos permite no repetir toda la información de una publicación (lo que sería necesario tener en cuenta al momento de actualizarla) y también soportar la información de la tabla Maestra donde cada publicación tenía una sola presentación.

Para insertar los datos de la tabla Maestra en el resto de las tablas decidimos usar sentencias de INSERT INTO con una query (SELECT) como data a insertar. Optamos por esta solución antes que el manejo de cursores.

En algunos casos nos tomamos ciertas libertades debido a los datos que se podían observar en la tabla Maestra. Un ejemplo de estos casos es los valores por default que utilizamos al migrar



datos como el medio de pago (siempre era “Efectivo”), el estado de las publicaciones (siempre “Publicada”), el rubro (siempre Otros) y el grado (siempre Bajo).

Esto permitió agilizar la migración sin tener que hacer joins para los cuales ya teníamos la información de antemano.