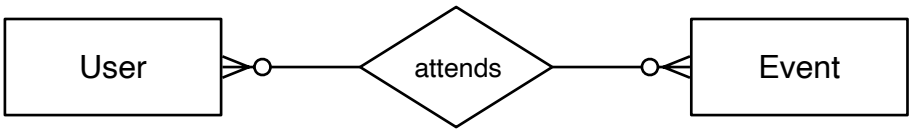
Eventos

Este ejercicio gira en torno a **la gestión de eventos** desarrollando un programa en **Python** mediante el paradigma de **programación orientada a objetos** junto con **acceso a datos**.

La idea es simular el **acceso a eventos** a través de una pequeña base de datos.

1. Base de datos

A continuación se presenta el **modelo entidad-relación** de la base de datos:



Se ha llevado a cabo la transformación a un **modelo relacional** y a continuación se muestran las tablas y columnas correspondientes de la base de datos:

Tabla `user`

Representa un usuario dentro del sistema de gestión de eventos.

Columna	Tipo	Descripción
`id`	Entero (PK)	Clave primaria
`username`	Texto (UNIQUE)	Nombre (alias) de usuario
`name`	Texto	Nombre de usuario
`surname`	Texto	Apellido

Ejemplo de registro:
`|2|'mhamilton'|'Margaret'|'Hamilton'|`

Tabla `event`

Representa un evento dentro del sistema de gestión de eventos.

Columna	Tipo	Descripción
`id`	Entero (PK)	Clave primaria
`name`	Texto (UNIQUE)	Nombre del evento
`seats`	Entero	Cantidad de sitios disponibles en el evento
`price`	Real	Precio del evento

Ejemplo de registro:
`|17|'Tomorrowland'|400000|27|`

Tabla `attendance`

Representa la asistencia de un usuario a un evento.

Columna	Tipo	Descripción
`id`	Entero (PK)	Clave primaria
`user_id`	Entero	Identificador del usuario
`event_id`	Entero	Identificador del evento
`num_tickets`	Entero	Número de entradas adquiridas

Recuerda añadir las restricciones SQL de *integridad referencial*:

```
FOREIGN KEY (user_id) REFERENCES user(id),
FOREIGN KEY (event_id) REFERENCES event(id),
```

Ejemplo de registro:
`|65|2|17|0|`

ATENCIÓN
Debes cumplimentar la siguiente función creando todas las tablas anteriormente descritas (conecta a la ruta `db_path`). La función se encuentra al principio de la plantilla...

```
def create_db(db_path: str = DB_PATH) -> None:
    ...
```

2. Clase `User`

Representa un usuario dentro del sistema de comercio electrónico.

La clase ya dispone de los siguientes atributos de **clase** (que puedes usar en cualquier método incluso con `self`):

- `con` ⇒ conexión a la base de datos (con factoría `Row`).
- `cur` ⇒ cursor a la base de datos.

Implementa, *al menos*, los siguientes **métodos**:

```
def __init__(self, username: str, name: str, surname: str, id: int = None):
```

- Constructor de la clase.
- Utiliza expresiones regulares para comprobar que el `username` siga estas reglas:
  - Empieza con una letra **minúscula**.
  - Termina con una letra **mayúscula**.
  - Está formado únicamente por letras, números y guiones bajos.
  - Tiene un mínimo de 10 caracteres.
- En caso de que no siga este patrón, debes elevar una excepción de tipo `ValueError` con el mensaje Username does not follow security rules
- Si todo ha ido bien, crea los mismos atributos que los parámetros pasados.

```
def save(self) -> None:
```

- Almacena el objeto actual `self` en la base de datos.
- Actualiza el atributo `id` con el identificador asignado en la tabla.

```
def update(self) -> None:
```

- Si el objeto aún no se ha guardado en la base de datos, lanza una excepción de tipo `ValueError` con el mensaje User has not been yet saved into DB
- No es necesario consultar la base de datos para saber si un usuario no se ha guardado en la base de datos. Piensa en el atributo `id` que tienes en el constructor y su valor por defecto.
- Si todo ha ido bien, actualiza todos los campos del objeto actual `self` en la base de datos utilizando el identificador como referencia.

```
def __str__(self):
```

- Representa el objecto actual en formato: `<name> <surname>`

```
@classmethod
def build_from_id(cls, user_id: int) -> User:
```

- Construye un objeto de tipo `User` a partir del identificador de usuario pasado por parámetros (mediante una consulta a la base de datos).
- Si el identificador de usuario no existe (en la base de datos) lanza una excepción de tipo `ValueError` con el mensaje User does not exist in DB

3. Clase `Event`

Representa un evento dentro del sistema de gestión de eventos.

La clase ya dispone de los siguientes atributos de **clase** (que puedes usar en cualquier método incluso con `self`):

- `con` ⇒ conexión a la base de datos (con factoría `Row`).
- `cur` ⇒ cursor a la base de datos.

Implementa, *al menos*, los siguientes **métodos**:

```
def __init__(self, name: str, seats: int, price: float, id: int = None):
```

- Constructor de la clase.
- Crea los mismos atributos que los parámetros pasados.

```
def save(self) -> None:
```

- Almacena el objeto actual `self` en la base de datos.
- Actualiza el atributo `id` con el identificador asignado en la tabla.

```
def update(self) -> None:
```

- Si el objeto aún no se ha guardado en la base de datos, lanza una excepción de tipo `ValueError` con el mensaje Event has not been yet saved into DB
- No es necesario consultar la base de datos para saber si un evento no se ha guardado en la base de datos. Piensa en el atributo `id` que tienes en el constructor y su valor por defecto.
- Si todo ha ido bien, actualiza todos los campos del objeto actual `self` en la base de datos utilizando el identificador como referencia.

```
def dispatch(self, num_tickets: int) -> None:
```

- Vende `num_tickets` entradas para el evento `self`.
- Si la cantidad a vender `num_tickets` es mayor que la *capacidad* del evento, lanza una excepción de tipo `ValueError` con el mensaje Not enough free seats for event
- Si todo ha ido bien, actualiza la cantidad de sitios *seats* del evento y actualiza la información del objeto en la base de datos.
- Haz uso del método `update()` implementado previamente.

```
def expand(self, seats: int) -> None:
```

- Aumenta el número de sitios para el evento `self`.
- Actualiza la *capacidad* del evento y actualiza la información del objeto en la base de datos.
- Haz uso del método `update()` implementado previamente.

```
def __repr__(self):
```

- El evento se representa por su nombre.

```
def __eq__(self, other: Event | object):
```

- Comprueba que dos eventos son iguales.
- Dos eventos son iguales si (y sólo si) sus nombres son iguales.
- Un evento es distinto a cualquier cosa que no sea un evento.

```
@classmethod
def build_from_id(cls, event_id: int) -> Event:
```

- Construye un objeto de tipo `Event` a partir del identificador de evento pasado por parámetros (mediante una consulta a la base de datos).
- Si el identificador de evento no existe lanza una excepción de tipo `ValueError` con el mensaje Event does not exist in DB

4. Clase `Attendance`

Representa la asistencia de un usuario a un evento.

La clase ya dispone de los siguientes atributos de **clase** (que puedes usar en cualquier método incluso con `self`):

- `con` ⇒ conexión a la base de datos (con factoría `Row`).
- `cur` ⇒ cursor a la base de datos.

Implementa, *al menos*, los siguientes **métodos**:

```
@classmethod
def purchase(cls, user_id: int, event_id: int, num_tickets: int) -> None:
```

- El usuario con identificador `user_id` compra `num_tickets` entradas para el evento con identificador `event_id`.
- Esto implica que hay que añadir una nueva fila en la tabla `attendance`.
- Además hay que actualizar los *sitios disponibles* del evento.
- Utiliza métodos ya definidos en las clases anteriores para ciertas partes de tu código.


```python
@classmethod
def cancel(cls, user_id: int, event_id: int) -> None:
```

- El usuario con identificador `user_id` cancela las entradas compradas para el evento con identificador `event_id`.
- Esto implica que hay que eliminar una fila en la tabla `attendance`.
- Además hay que actualizar los *sitios disponibles* del evento. Ten en cuenta que el número de tickets adquiridos está en la tabla `attendance`.
- Utiliza métodos ya definidos en las clases anteriores para ciertas partes de tu código.

