MAKE SCHOOL

SQLAlchemy & Models

BEW 1.2

Agenda

- Learning Outcomes
- Warm-Up: My Favorite Mistake
- Lab Activity - Pair Programming
- Types of Fields
- **BREAK**
- One-to-Many Relationships
- Many-to-Many Relationships
- Lab Activity - Pair Programming
- Wrap-Up

Learning Outcomes

By the end of today, you should be able to...

1. **Write** models containing one-to-many and many-to-many relationships using SQLAlchemy.
2. **Create, read, update, and delete** data from a SQLAlchemy database using model classes.

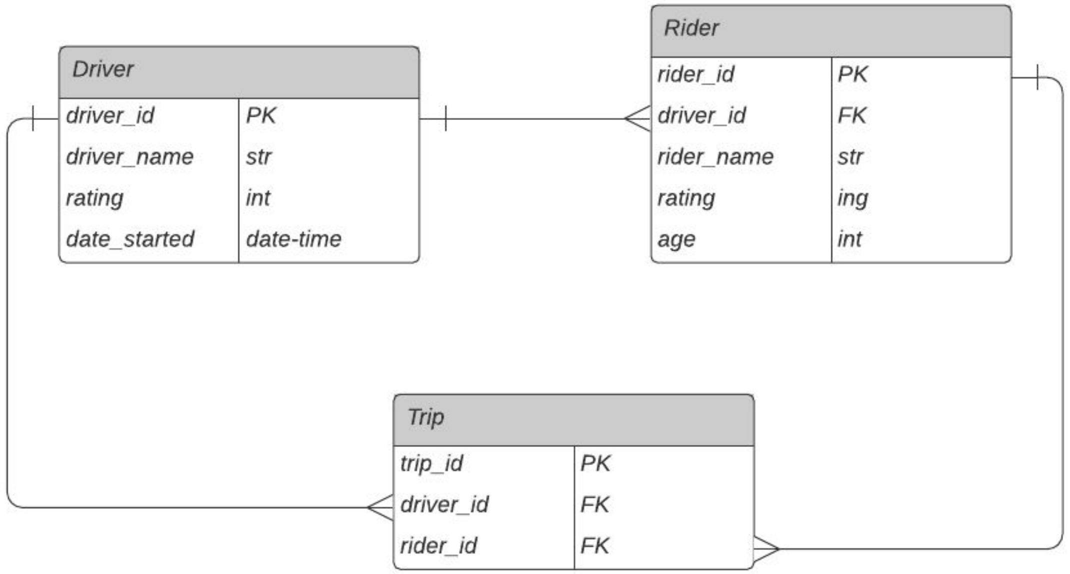
Warm-Up - My Favorite Mistake (5 minutes)

In a group of 3, discuss each of the following **3 submissions** and answer:

1. What did this person do **right**?
2. What **misconception** did this person have?

Go to the next slides (3, 4, 5) to view the submissions.

"Lyft" ERD

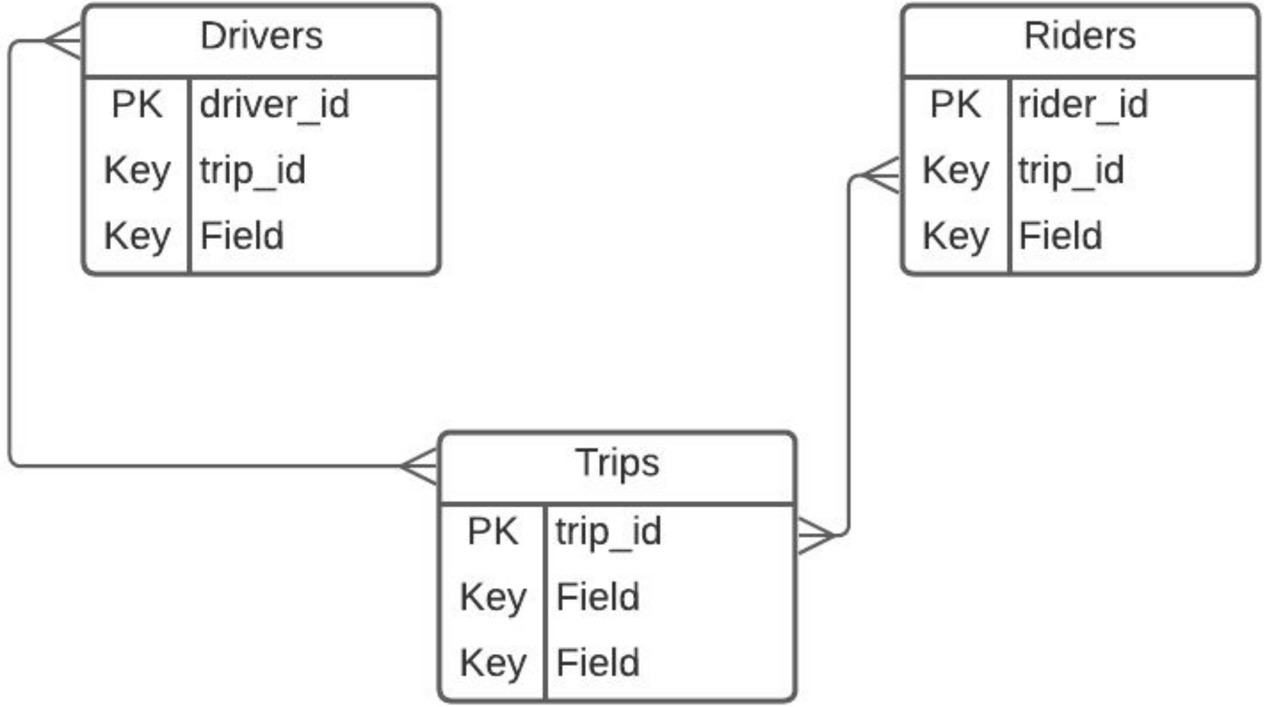


Driver	
driver_id	PK
driver_name	str
rating	int
date_started	date-time

Rider	
rider_id	PK
driver_id	FK
rider_name	str
rating	ing
age	int

Trip	
trip_id	PK
driver_id	FK
rider_id	FK

"Lyft" ERD


Drivers
PK driver_id
Key trip_id
Key Field
Riders
PK rider_id
Key trip_id
Key Field
Trips
PK trip_id
Key Field
Key Field

SQL Query

SQL code to select all songs on albums published between 1970 and 1980:

```
SELECT Songs.title
FROM Albums, Songs
WHERE year_published BETWEEN 1970 AND 1980;
```

A note on this class: Desirable difficulty

Often during this course, your first exposure to the topic being introduced will be via a **lab activity** instead of via direct instruction.

You may find these activities to be difficult or confusing, or that they take you out of your comfort zone.

That's okay - and expected! ***If a topic is confusing at first, and you are able to resolve that confusion, you will learn it better than if you were just given the answer.***

For more info on desirable difficulties, see this Stanford article.

MAKE SCHOOL

What are Models?

What are Models?

A **model** is a special Python class that defines a database table.

In this class we'll be using the **Flask-SQLAlchemy** library to define our models.

This will make it much easier for us to create and query our data.

Activity (25 minutes)

Complete Part 1 of the SQLAlchemy Models Lab with a partner.

Be sure to practice good pair programming!

Models

Here is an example of a model class:

```
class Author(db.Model):
    """Author model."""
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(80), nullable=False)
    books = db.relationship('Book', back_populates='author')
```

Types of Columns

In addition to primary & foreign key, models can have the following field types:

Source

`Integer`	an integer
`String(size)`	a string with a maximum length (optional in some databases, e.g. PostgreSQL)
`Text`	some longer unicode text
`DateTime`	date and time expressed as Python **datetime** object.
`Float`	stores floating point values
`Boolean`	stores a boolean value
`Enum`	an enum value

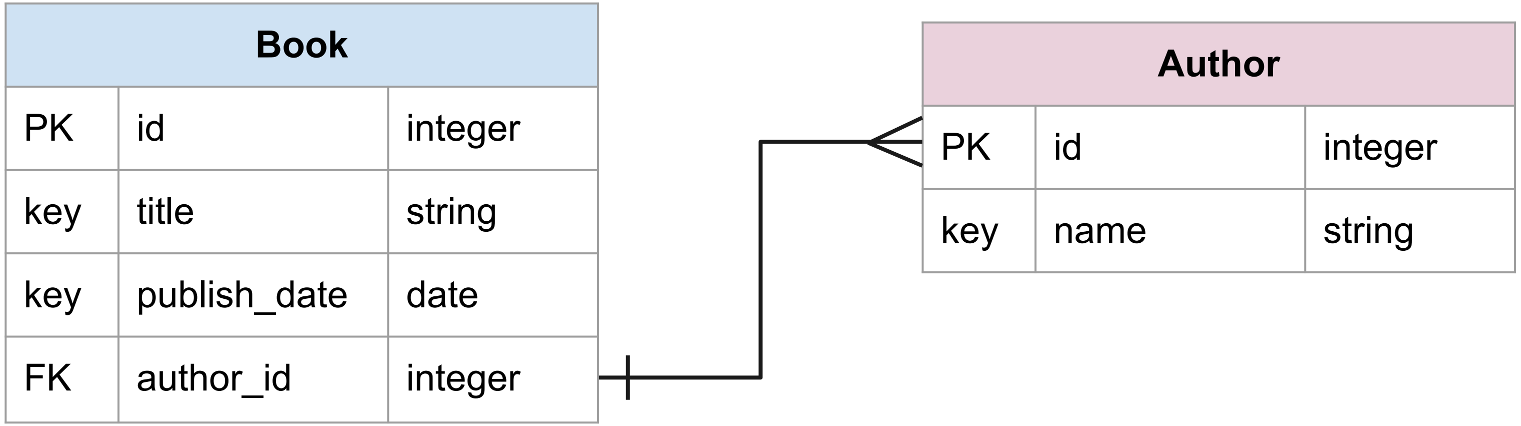
MAKE SCHOOL

Break - 10 min

One-to-Many Relationships

One-to-Many Relationships

Let's say our site has **Book** and **Author** entities with the following relationship:



Book		
PK	id	integer
key	title	string
key	publish_date	date
FK	author_id	integer

Author		
PK	id	integer
key	name	string

One-to-Many Relationships

We can use a **db.ForeignKey** field to store the author id:

```
class Author(db.Model):
   id = db.Column(db.Integer, primary_key=True)
   name = db.Column(db.String(80), nullable=False)

class Book(db.Model):
   id = db.Column(db.Integer, primary_key=True)
   title = db.Column(db.String(80), nullable=False)
   publish_date = db.Column(db.Date)
   author_id = db.Column(db.Integer, db.ForeignKey('author.id'))
```

One-to-Many Relationships

However, it would be pretty annoying to search for a book, and then only be able to see its author's id.

Ideally, when we query for a book, **we want to get the author's info** without having to do an extra query.

Luckily, SQLAlchemy has us covered! :)

One-to-Many Relationships

We can use **relationship** fields to tell SQLAlchemy to automatically populate some extra data when it does a query. These aren't real columns in the SQL table, unlike the **Column** fields.

```
class Author(db.Model):
    # ...
    books = db.relationship('Book', back_populates='author')

class Book(db.Model):
    # ...
    author_id = db.Column(db.Integer, db.ForeignKey('author.id'))
    author = db.relationship('Author', back_populates='books')
```

One-to-Many Relationships

When two models both have relationships to each other, we can use **back_populates** to denote a two-way relationship.

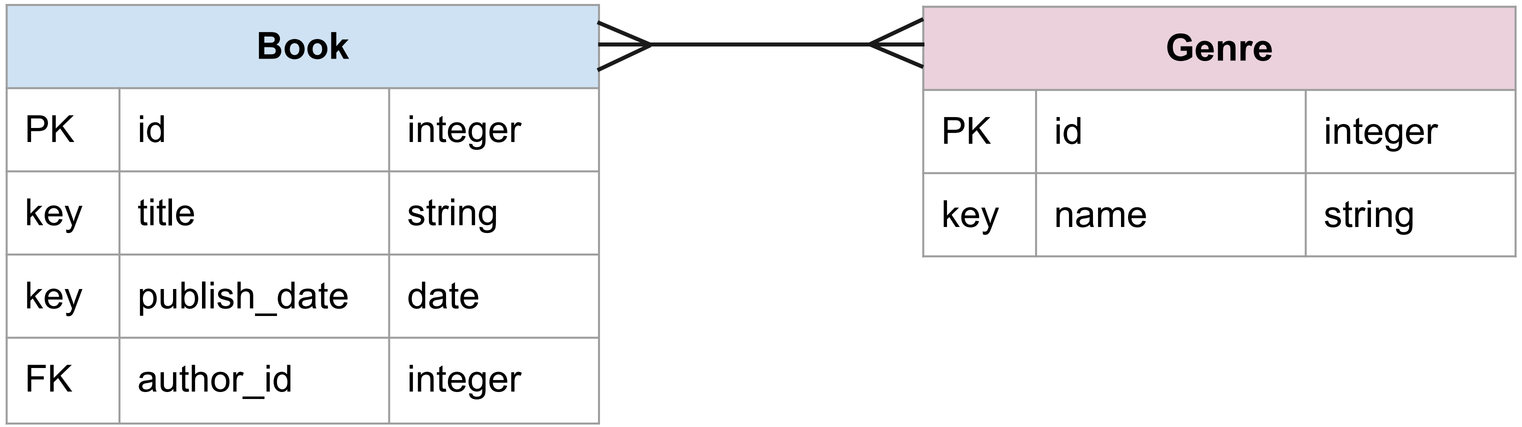
```
class Author(db.Model):
    # ...
    books = db.relationship('Book', back_populates='author')

class Book(db.Model):
    # ...
    author_id = db.Column(db.Integer, db.ForeignKey('author.id'))
    author = db.relationship('Author', back_populates='books')
```

MAKE SCHOOL

Many-to-Many Relationships

Many-to-Many Relationships

Now let's say we have the following relationship:



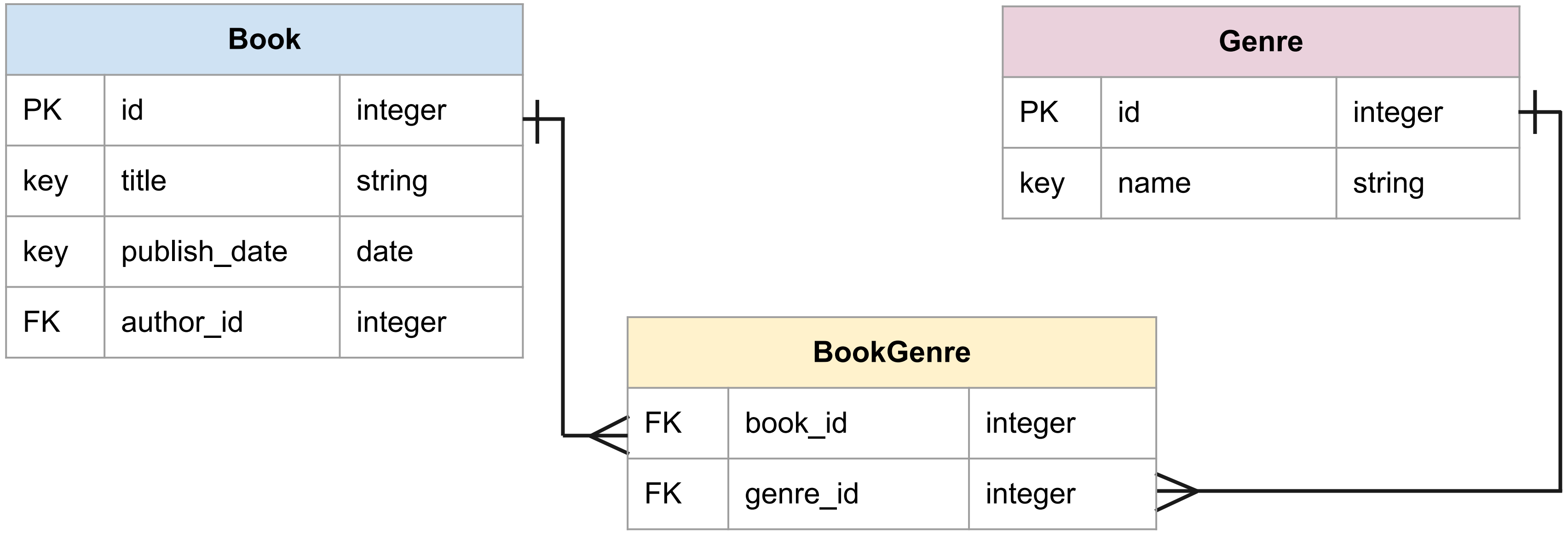
Book		
PK	id	integer
key	title	string
key	publish_date	date
FK	author_id	integer

Genre		
PK	id	integer
key	name	string

Notice that neither **Book** nor **Genre** have a foreign key field that explicitly refers to the other.

Many-to-Many Relationships

Remember, **we can re-write this relationship** by using a **bridge table**:



Book		
PK	id	integer
key	title	string
key	publish_date	date
FK	author_id	integer

Genre		
PK	id	integer
key	name	string

BookGenre		
FK	book_id	integer
FK	genre_id	integer

Notice that there isn't any important information in the bridge table. **It's just a tool** to get us that many-to-many relationship.

Many-to-Many Relationships

It's recommended to make the bridge table a **db.Table**, not a **db.Model**,

because we don't need to query it directly.

```
class Genre(db.Model):
    # ...
    books = db.relationship('Book', secondary='book_genre', back_populates='genres')

class Book(db.Model):
    # ...
    genres = db.relationship('Genre', secondary='book_genre', back_populates='books')

book_genre_table = db.Table('book_genre',
    db.Column('book_id', db.Integer, db.ForeignKey('book.id')),
    db.Column('genre_id', db.Integer, db.ForeignKey('genre.id'))
)
```

Activity (25 minutes)

Complete Parts 2 and 3 of the SQLAlchemy Models Lab with a partner.

Wrap-Up

Wrap-Up

Finish lab activity (parts 2 and 3)

Start on Homework 2: Events Site