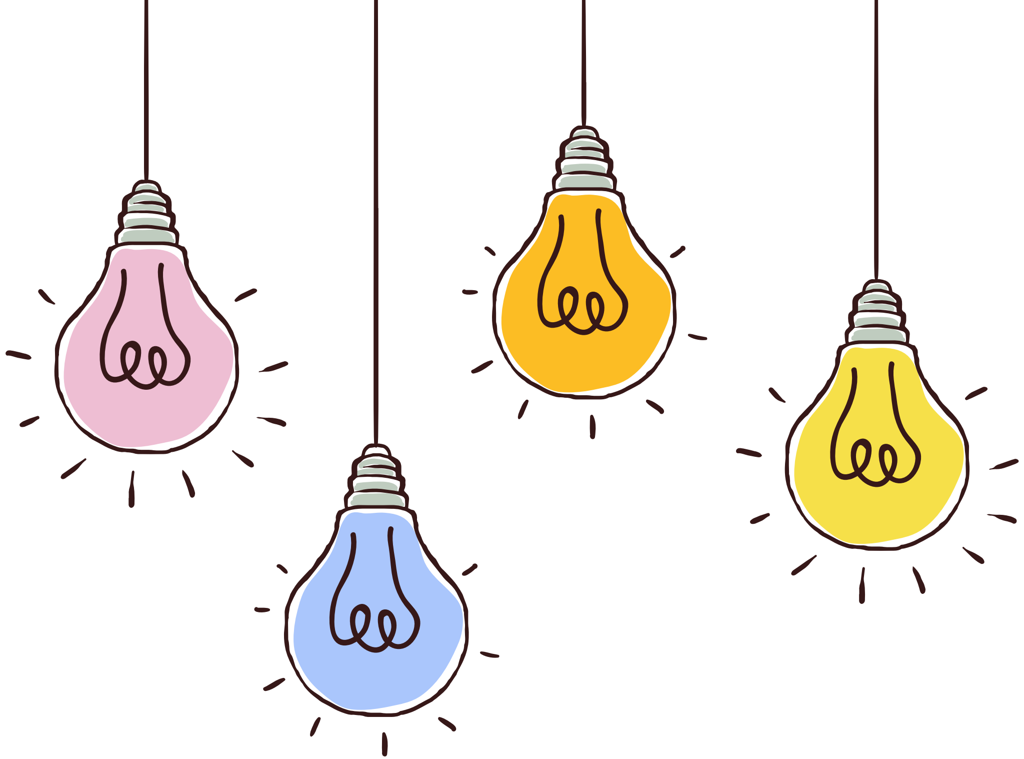
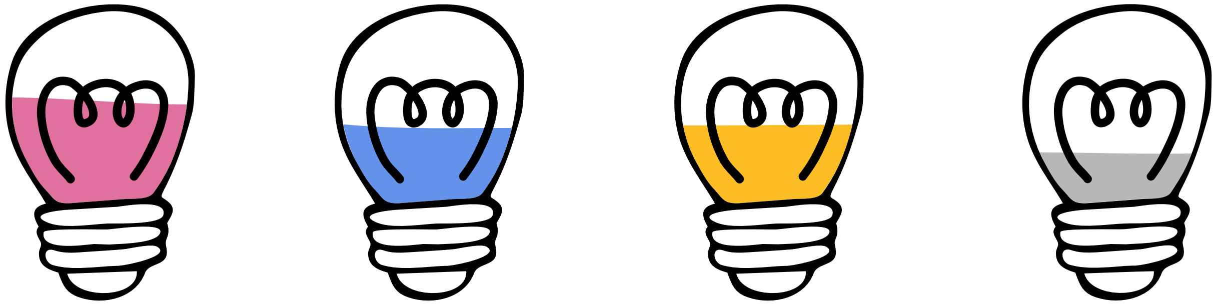


SQL Case Study: Simple Instagram Clone

Kelsey Flynn

Project Stages



01

Planning Schema

02

Create tables in GoormIDE

03

Fill table data

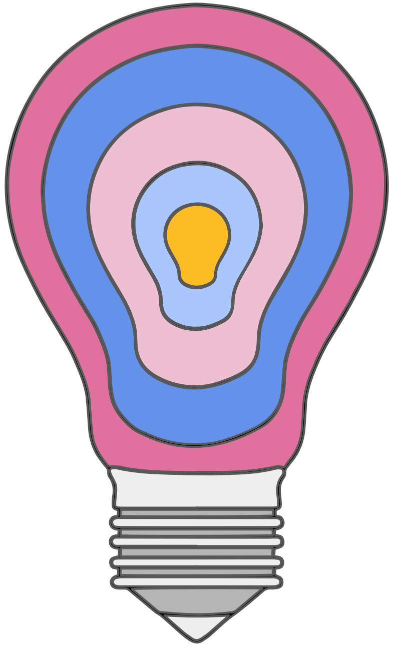
04

SQL Queries

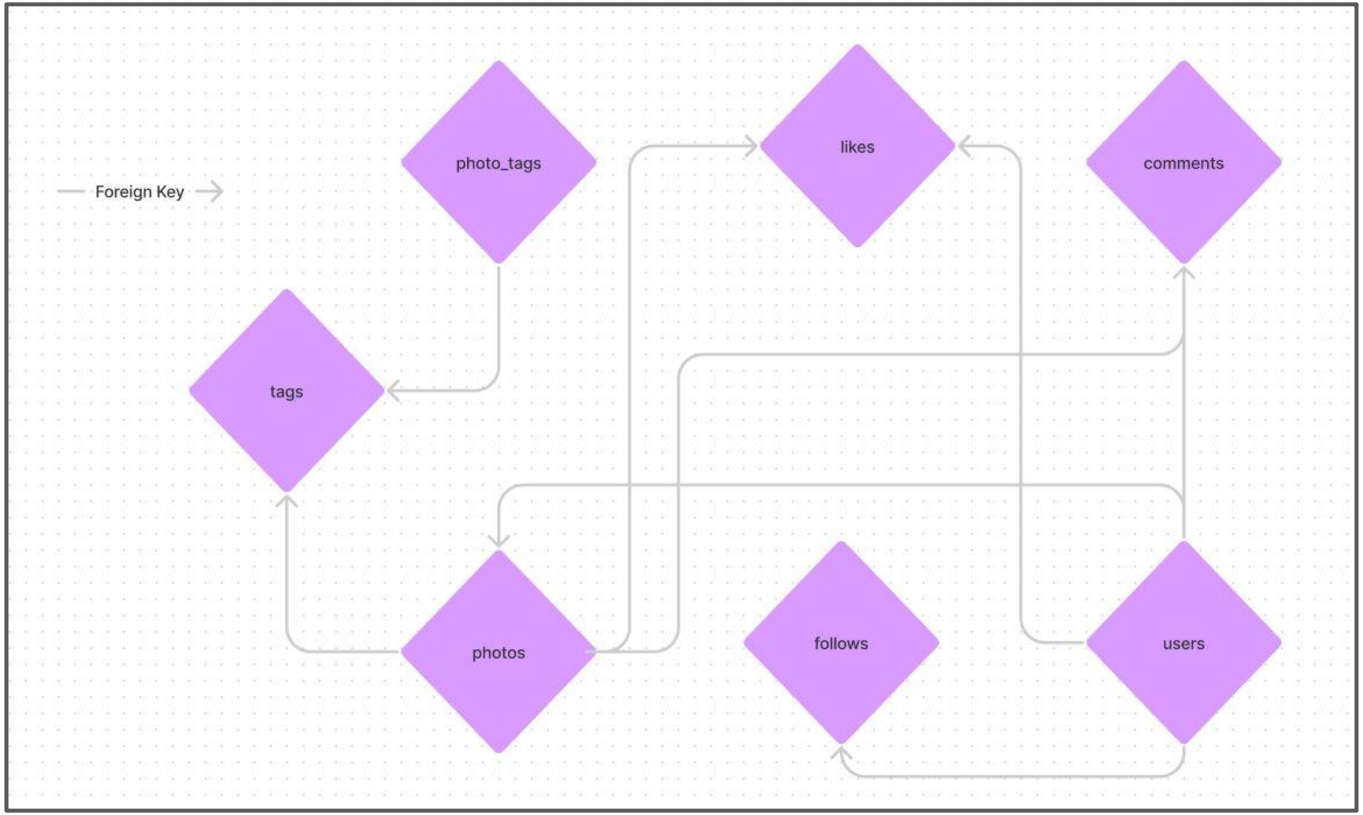
Planning Schema

Following a logical order:

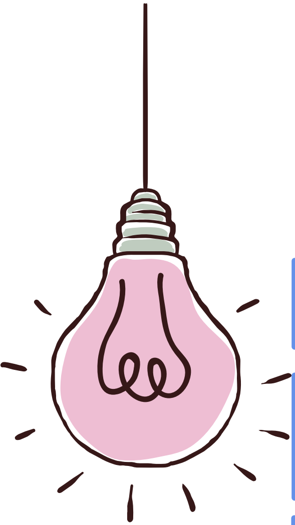
The “leading” table, contains no foreign keys.	**users**
Users post photos, will need to reference users.	**photos**
Users comment on photos, foreign keys for users & photos.	**comments**
Users like photos, same foreign keys as comments.	**likes**
Users can follow others and be followed, references users 2x.	**follows**
Photos can be associated with tags, but no foreign keys.	**tags**
To see which photos have a certain tag, will need to reference “parent” tables for foreign keys.	**photo_tags**

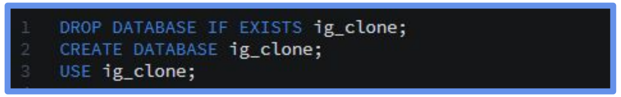


Planning Schema

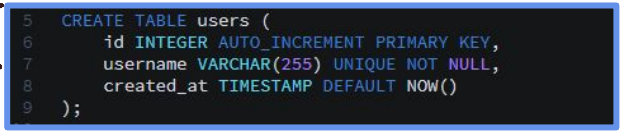


Creating Tables in GoormIDE

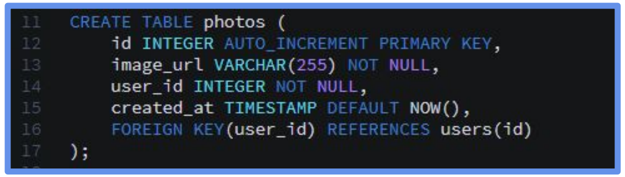




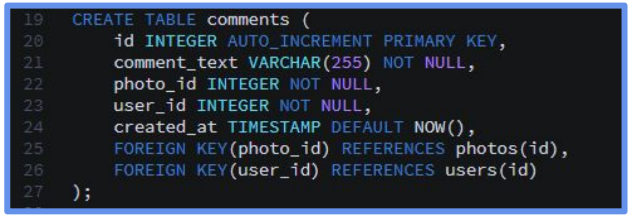
```sql
DROP DATABASE IF EXISTS ig_clone;
CREATE DATABASE ig_clone;
USE ig_clone;
```



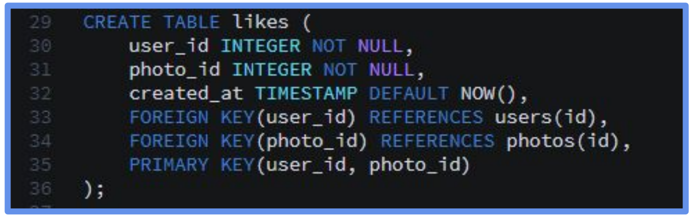
```sql
CREATE TABLE users (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);
```



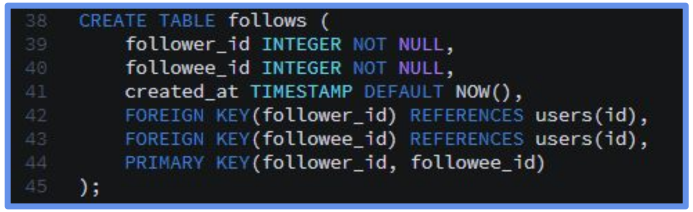
```sql
CREATE TABLE photos (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    image_url VARCHAR(255) NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(user_id) REFERENCES users(id)
);
```



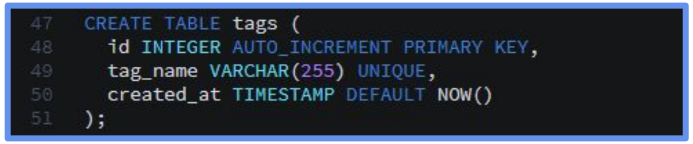
```sql
CREATE TABLE comments (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    comment_text VARCHAR(255) NOT NULL,
    photo_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(photo_id) REFERENCES photos(id),
    FOREIGN KEY(user_id) REFERENCES users(id)
);
```



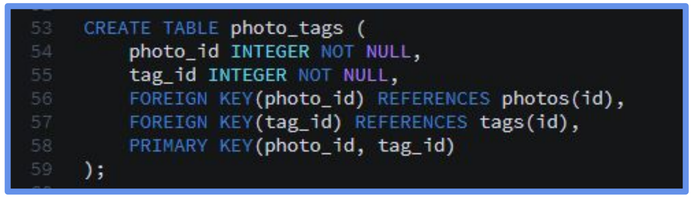
```sql
CREATE TABLE likes (
    user_id INTEGER NOT NULL,
    photo_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(user_id) REFERENCES users(id),
    FOREIGN KEY(photo_id) REFERENCES photos(id),
    PRIMARY KEY(user_id, photo_id)
);
```



```sql
CREATE TABLE follows (
    follower_id INTEGER NOT NULL,
    followee_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    FOREIGN KEY(follower_id) REFERENCES users(id),
    FOREIGN KEY(followee_id) REFERENCES users(id),
    PRIMARY KEY(follower_id, followee_id)
);
```



```sql
CREATE TABLE tags (
  id INTEGER AUTO_INCREMENT PRIMARY KEY,
  tag_name VARCHAR(255) UNIQUE,
  created_at TIMESTAMP DEFAULT NOW()
);
```



```sql
CREATE TABLE photo_tags (
    photo_id INTEGER NOT NULL,
    tag_id INTEGER NOT NULL,
    FOREIGN KEY(photo_id) REFERENCES photos(id),
    FOREIGN KEY(tag_id) REFERENCES tags(id),
    PRIMARY KEY(photo_id, tag_id)
);
```

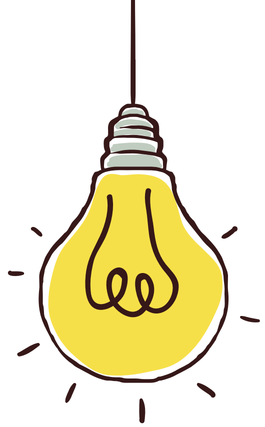
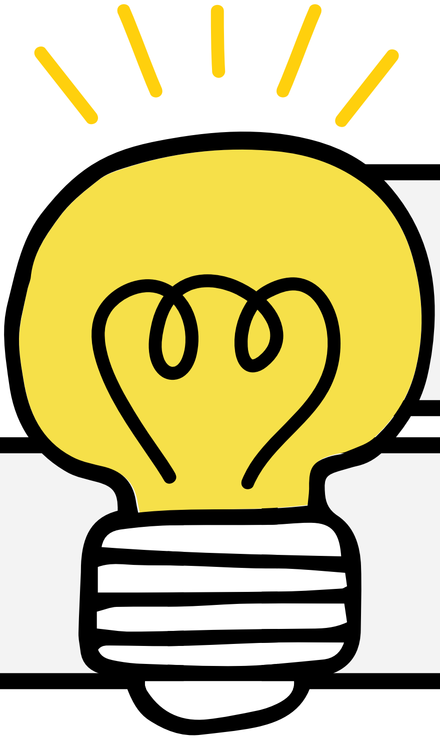


Table Data



Data provided by Colt Steele, Developer & Bootcamp Instructor

- 100 users
- 7488 comments
- 7623 follows
- 8782 likes

- 257 photos
- 21 tags
- 501 photo_tags

Planning SQL Queries: What do we want to know?

We want to find the best & most loyal customers. Would give insights to send a thank you post or a sponsorship!

We want to find the best time to create marketing plans for certain times of the week/year and tag popularity!

We want to find the "dead" accounts, to verify a fake account or to send another welcome/how to email to users!

SQL Queries Pseudo Code & Overview

Find the 5 oldest users

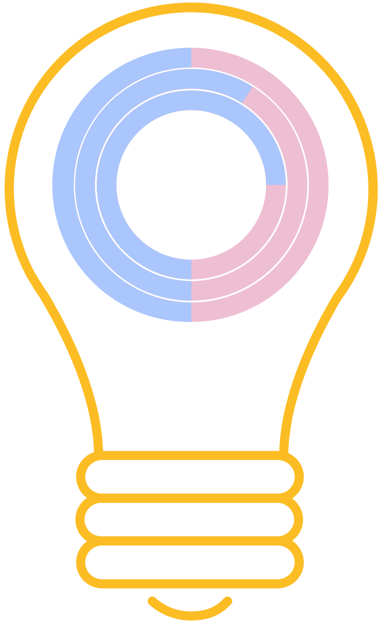
Find the most popular registration date

Find the most popular hashtags

Identify most popular photo (and user who created it)

Find inactive users

Find those bots!



Queries

```
SELECT username
FROM users
LEFT JOIN photos
ON users.id = photos.user_id
WHERE photos.id IS NULL;
```

```
SELECT username, photos.id, photos.image_url, COUNT(*) as total
FROM photos
INNER JOIN likes
ON likes.photo_id = photos.id
INNER JOIN users ON photos.user_id = users.id
GROUP BY photos.id
ORDER BY total DESC
LIMIT 1;
```

```
SELECT
        DAYNAME(created_at) AS day,
        COUNT(*) AS total
FROM users
GROUP BY day
ORDER BY total DESC
LIMIT 2;
```

```
SELECT
        Tags.tag_name,
        COUNT(*) AS total
FROM photo_tags
JOIN tags
ON photo_tags.tag_id = tags.id
GROUP BY tags.id
ORDER BY total DESC
LIMIT 5;
```

```
SELECT
        Username,
        COUNT(*) AS num_likes
FROM users
INNER JOIN likes
ON users.id = likes.user_id
GROUP  BY likes.user_id
HAVING num_likes =
        (SELECT
        Count(*)
        FROM   photos);
```

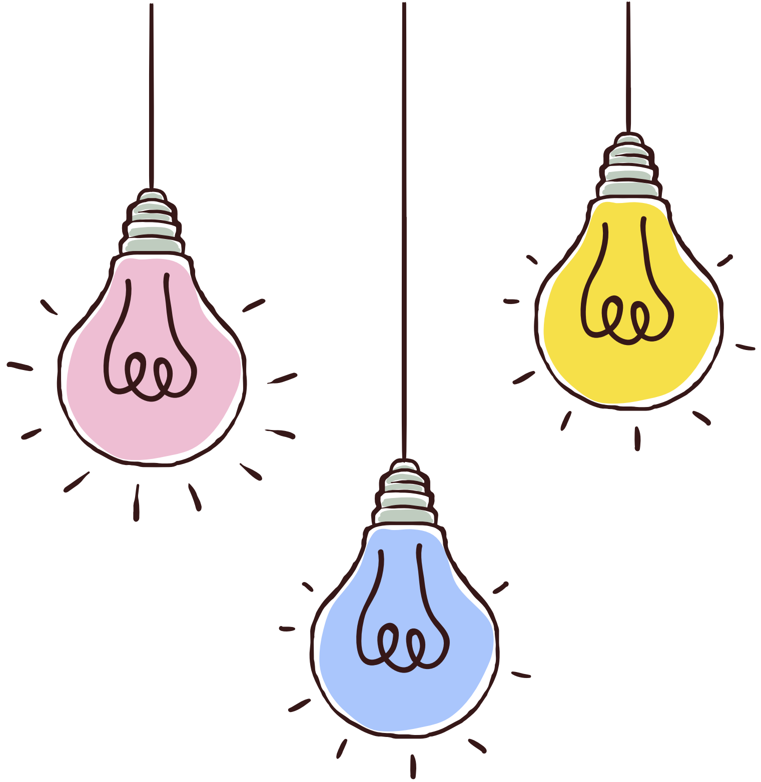
```
SELECT *
FROM users
ORDER BY created_at
lIMIT 5;
```

Takeaways

1. Best practices for loading and working with large amounts of data.

2. INNER vs. LEFT vs. RIGHT JOIN experience.

3. How to “slim down” a large concept like Instagram into smaller pieces for a mock structure.



Thank you to slidego.com for the resources!