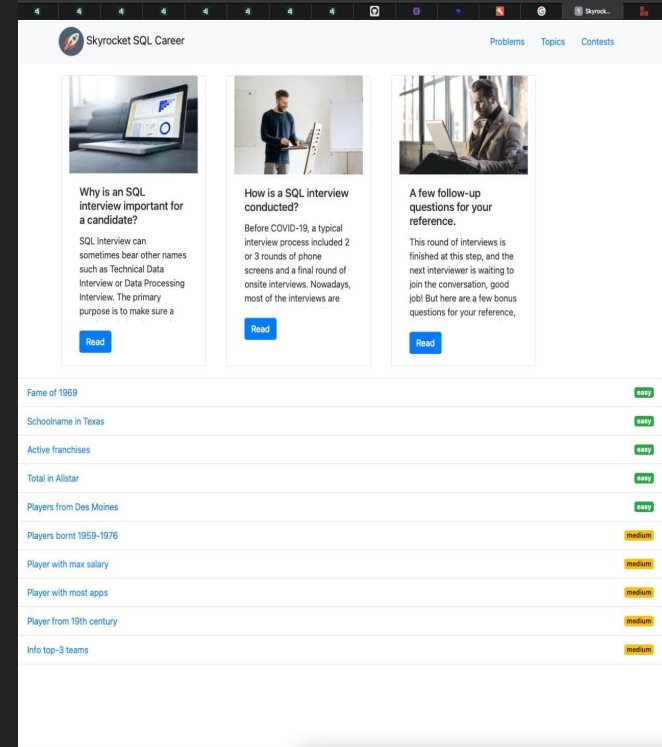


# SkyrocketSQLCareer

Team: Uzbekistan

# General information about our project

SkyrocketSQLCareer is a website where people can prepare for the upcoming tech interviews. Usually Data Analysts, Security specialists, Database managers, Data Scientists that are applying for the top tech companies facing a technical interview where their need to solve SQL puzzles, write complex SQL queries, or explain how relational database work under the hood, relational algebra e.t.c. to prove their competences. We want to help candidates with their interviews by providing a platform which has an everything in one place.



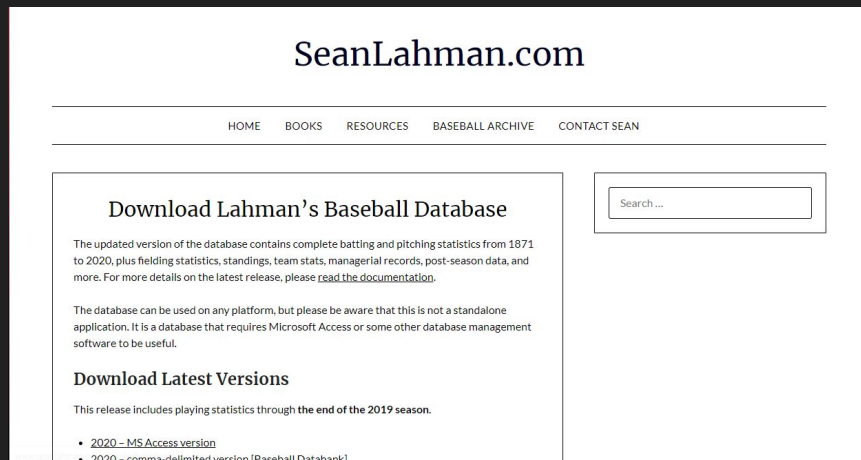
# Our goals:

We wanted to provide:

- SQL puzzles where users can write a query and see if they are right.
- Complex and simple SQL query questions for checking users understanding of certain SQL topics.
- Documentation about SQL topics, interview preparation and e.t.c.

# Dataset

Since our aim was to provide platform for educational purposes, we decided to explore for a huge database which consist a lot of data and tuples. Lahman's Baseball dataset was a perfect choice, since it satisfied all requirements that we wanted to see in our project.



# Stats of our dataset

1. 168k	11. 6.7k	21. 6.7k
2. 100k	12. 6k	22. 6k
3. 99k	13-14. 5k	23. 5k
4. 43.3k	13-14. 5k	24. 5k
5. 24.8k	15. 4.1k	
6. 18.6k	16. 3.4k	
7. 17.3k	17. 2.8k	
8. 12k	18. 1.2k	
9. 11.9k	19. 402	
10. 11.3k	20. 300	Total: 24 relations, 541,550 tuples

# Database


We used PostgreSQL database system to write queries. We faced a lot of problems: installing data, importing them to database, then when we migrate existing database with postgresQL Django project. To solve problems with migrating, we just used Django helpers and wrote code around SQLite database.



# How it works?

We store in our database only the right query, and backend used to receive SQLite answer dialect. And when user pushes 'Submit', we are catching submitted SQL answer, using inner python tools we are trying to run raw sql on our database, then we are running correct sql answer stored for the question, and comparing the answer strings, if they match user had submitted the correct answer, if not, he had made a mistake. When user press 'create contest', then he chooses the questions from the list that contest contains of, and after we create the contest record and generate unique link by which users can connect to the contest and solve problems while contest session exists, there is no any Auth system in the project, so basically anyone can join.

# How it works?

 Skyrocket SQL Career

Problems Topics Contests

## Fame of 1969

easy


Find all hall of fame players that played there in 1969. Table name: halloffame, attribute name: yearid

Solution:

select \* from halloffame where yearid = "

Submit

Error in query

 Skyrocket SQL Career

Problems Topics Contests

## Fame of 1969

easy

Find all hall of fame players that played there in 1969. Table name: halloffame, attribute name: yearid

Solution:

select \* from halloffame where yearid = '1969'

Submit

Compiled Successfully!

Naive implementation of our SQL compiler:

- We have a Model that has following attributes:

```
class Question(models.Model):
    text = models.CharField(max_length=1000)
    title = models.CharField(max_length=100)
    difficulty = models.CharField(max_length=10)
    solution = models.CharField(max_length=1000)

    def checkSolution(self, data_to_check):
        with connection.cursor() as cursor:
            try:
                cursor.execute(self.solution)
                data = cursor.fetchall()
                return data == data_to_check
            except (KeyError):
                return False
```

- Where solution is the SQL query that is used for checking for answers.
- User submits the SQL query and we are checking from incoming post request if solution is correct:

```
if request.method == "POST":
    form = ProblemSolution(request.POST)
    if form.is_valid():
        ans = form.cleaned_data["solution"]
        with connection.cursor() as cursor:
            try:
                cursor.execute(ans)
                check = cursor.fetchall()
                if question.checkSolution(check):
                    return render(request, 'skyrocketSql/problem.html', {
                        'question': question,
                        'form': form,
                        'isCorrect': "correct"
                    })
            else:
                return render(request, 'skyrocketSql/problem.html', {
                    'question': question,
                    'form': form,
                    'isCorrect': "incorrect"
                })
```



## Challenges during implementation

1. During of making E/R diagram, we faced with problem that caused disorder and messiness in our diagram. The reason of this was the amount of relations in our dataset(which was 24).
2. Since some of team members had “not the most powerful” devices(laptops/PCs), we faced with problem - lack of performance during running our SQL queries, since we have some table with large amount of tuple(150-200k).

# Failures during implementation

- We didn't consider the complexity of our SQL queries during the Phase 3, when we started to create queries, and in Phase 8, when we tried to calculate true cardinality and query execution times, amount of these queries weren't enough to satisfy the requirements in the proposal(we've got only 4 out of 7 required queries).
- Some problems with time management during doing phases(we've lost most points because of late submission).

Thank you for attention!!!