Predicting the Centennial Conference Track & Field Championships

By Mirra Klimov (mklimov1@jh.edu) (601.315) & Spencer Ye (sye15@jh.edu) (601.415)

GitHub Link: github.com/ye-spencer/Databases-Final-Project
Website: https://cctf-prediction.vercel.app/

Our project revolves around track and field data from the Centennial Conference. We collected current and historical performance data, and created a modern interface to it, while adding analysis and ultimately predicting the results of the championship meet.

Introduction

Both of us are sprinters on the Johns Hopkins Track and Field team, with Mirra doing short sprints (60m/100m/200m) and Spencer doing long sprints (200m/400m). Every year, members of our team try to predict the results of the Centennial Conference Championship meet manually with pen and paper. This was a perfect opportunity to apply our database knowledge, and so our project was born.

Project Overview

Our project has two main parts. First, we had results viewing. All collegiate track and field data must be uploaded to the Track & Field Results Reporting System (TFRRS). However, TFRRS is notoriously slow. Rebuilding a TFRRS-like system using modern frameworks has allowed us to reduce loading times of basic stats from seconds to milliseconds. Second, we have result predictions. Using historical performances, we are able to predict athletes' performances on championship Sunday and calculate the team scores. See the user guide for more specifics on how to use the website and other parts.

Our repository is split into four subfolders. The scrape_tffrs folder has all the files used to scrape TFRRS for the bulk of our data. The db_generating folder has all the files used to create our tables and populate all our manual data. The predict-the-centenni-podium folder has all the files used in our website. The prediction_analysis folder has all the files used to do extra analysis on our data.

Database Design

Schema

Our database has 10 main tables, shown below. This decomposition of our fields led to the least amount of data being stored, that is, the most normalized form. It also enables handling edge cases, like transferring students within the Centennial Conference.

GeographicLocation: LocationID, CityName, StateCode, CountryCode
School: SchoolID, SchoolName, StreetAddress, EnrollmentSize, HasIndoorFacility, LocationID (to GeographicLocation)
TrackMeet: MeetID, MeetName, StartDate, EndDate
Athlete: AthleteID, AthleteLastName, AthleteFirstName, Gender
AthleteSeason: AthleteSeasonID, SeasonType, SeasonYear, ClassYear, AthleteID (to Athlete), SchoolID (to School)
TrackEvent: EventID, EventName, EventType, MeasureUnit, IsRelay

RelayTeam: RelayTeamID, SchoolID (to School), EventID (to TrackEvent), MeetID (to TrackMeet)
RelayTeamMembers: RelayTeamID (to RelayTeam), AthleteSeasonID (to AthleteSeason), LegNum
Performance: PerformanceID, ResultValue, WindGauge, MeetID (to TrackMeet), EventID (to TrackEvent), AthleteSeasonID (to AthleteSeason), RelayTeamID (to RelayTeam)
CentennialConferenceEvents: EventID, Indoor, Outdoor

The full relational table specification in the SQL Database Definition Language is pasted below, along with example values for each field.

```
DROP TABLE IF EXISTS GeographicLocation CASCADE;
CREATE TABLE GeographicLocation (
    LocationID      SERIAL PRIMARY KEY, -- 1, 10,
    CityName        VARCHAR(100) NOT NULL, -- Baltimore, New York
    StateCode       VARCHAR(2) NOT NULL, -- MD, NY
    CountryCode     VARCHAR(2) NOT NULL -- US
);

DROP TABLE IF EXISTS School CASCADE;
CREATE TABLE School (
    SchoolID        VARCHAR(20) PRIMARY KEY, -- Johns_Hopkins, Ursinus
    SchoolName      VARCHAR(100) NOT NULL, -- Johns Hopkins University, Ursinus College
    LocationID      INT NOT NULL REFERENCES GeographicLocation(LocationID), -- 1, 101
    StreetAddress   VARCHAR(100) NOT NULL, -- 3400 North Charles Street, 101 West End Avenue
    EnrollmentSize  INT NOT NULL, -- 10000, 1000
    HasIndoorFacility BOOLEAN NOT NULL -- True, False
);

DROP TABLE IF EXISTS TrackMeet CASCADE;
CREATE TABLE TrackMeet (
    MeetID          INT PRIMARY KEY, -- 1, 101 (TFRRS Meet ID)
    MeetName        VARCHAR(200) NOT NULL, -- Navy Indoor Invitational, Black and Blue
Invitational
    StartDate       DATE NOT NULL, -- 2025-12-07, 2025-12-08
    EndDate         DATE NOT NULL -- 2025-12-07, 2025-2026
);

DROP TABLE IF EXISTS Athlete CASCADE;
CREATE TABLE Athlete (
    AthleteID       INT PRIMARY KEY, -- 1, 101 (TFRRS Athlete ID)
    AthleteLastName VARCHAR(100) NOT NULL, -- Ye, Klimov
    AthleteFirstName VARCHAR(100) NOT NULL, -- Spencer, Mirra
    Gender          VARCHAR(1) NOT NULL CHECK (Gender IN ('M', 'F')) -- M, F
);

DROP TABLE IF EXISTS AthleteSeason CASCADE;
CREATE TABLE AthleteSeason (
    AthleteSeasonID SERIAL PRIMARY KEY, -- 1, 101
    AthleteID       INT NOT NULL REFERENCES Athlete(AthleteID), -- 1, 101
    SchoolID        VARCHAR(20) NOT NULL REFERENCES School(SchoolID), -- Johns_Hopkins
```

```
    SeasonType      VARCHAR(10) NOT NULL CHECK (SeasonType IN ('Indoor', 'Outdoor')), --
Indoor, Outdoor
    SeasonYear      INT NOT NULL, -- 2025, 2024
    ClassYear       VARCHAR(2) NOT NULL CHECK (ClassYear IN ('FR', 'SO', 'JR', 'SR')), -- FR,
SO, JR, SR
    UNIQUE (AthleteID, SeasonType, SeasonYear)
);

DROP TABLE IF EXISTS TrackEvent CASCADE;
CREATE TABLE TrackEvent (
    EventID         INT PRIMARY KEY, -- 1, 101 (TFRRS Event ID)
    EventName       VARCHAR(20) NOT NULL, -- 100m, 4x100m, Discus
    EventType       VARCHAR(8) NOT NULL CHECK (EventType IN ('sprints', 'distance', 'jumps',
'throws', 'combined')), -- sprints, distance, jumps, throws, combined
    MeasureUnit     VARCHAR(7) NOT NULL CHECK (MeasureUnit IN ('seconds', 'meters',
'points')), -- seconds, meters, points
    IsRelay         BOOLEAN NOT NULL -- True, False
);

-- NOTE: Not all relay teams have known team members
DROP TABLE IF EXISTS RelayTeam CASCADE;
CREATE TABLE RelayTeam (
    RelayTeamID     SERIAL PRIMARY KEY, -- 1, 101
    SchoolID        VARCHAR(20) NOT NULL REFERENCES School(SchoolID), -- Johns_Hopkins,
Ursinus
    EventID         INT NOT NULL REFERENCES TrackEvent(EventID), -- 1, 101
    MeetID          INT NOT NULL REFERENCES TrackMeet(MeetID) -- 1, 101
);

DROP TABLE IF EXISTS RelayTeamMembers CASCADE;
CREATE TABLE RelayTeamMembers (
    RelayTeamID       INT NOT NULL REFERENCES RelayTeam(RelayTeamID), -- 1, 101
    AthleteSeasonID   INT NOT NULL REFERENCES AthleteSeason(AthleteSeasonID), -- 1, 101
    LegNum            INT NOT NULL CHECK (LegNum IN (1, 2, 3, 4)), -- 1, 2, 3, 4
    PRIMARY KEY (RelayTeamID, AthleteSeasonID)
);

DROP TABLE IF EXISTS Performance CASCADE;
CREATE TABLE Performance (
    PerformanceID    SERIAL PRIMARY KEY, -- 1, 101
    MeetID          INT NOT NULL REFERENCES TrackMeet(MeetID), -- 1, 101
    EventID         INT NOT NULL REFERENCES TrackEvent(EventID), -- 1, 101
    AthleteSeasonID  INT REFERENCES AthleteSeason(AthleteSeasonID), -- 1, 101
    RelayTeamID      INT REFERENCES RelayTeam(RelayTeamID), -- 1, 101
    ResultValue      DECIMAL(8, 2) NOT NULL, -- 8394, 10.12, 1:52.12
    WindGauge        DECIMAL(3, 1), -- 0.0, 2.1
    CHECK ((RelayTeamID IS NULL AND AthleteSeasonID IS NOT NULL) OR (AthleteSeasonID IS NULL
AND RelayTeamID IS NOT NULL))
);

DROP TABLE IF EXISTS CentennialConferenceEvents CASCADE;
CREATE TABLE CentennialConferenceEvents (
    EventID         INT PRIMARY KEY, -- 1, 101 (TFRRS Event ID)
    Indoor          BOOLEAN NOT NULL, -- True, False
    Outdoor         BOOLEAN NOT NULL -- True, False
);
```

Database Hosting

Instead of using the MySQL instance on dbase.cs.jhu.edu, we opted to utilize Neon (neon.com), a hosted, serverless PostgreSQL service. We had used Neon in previous projects and had enjoyed the generous free tier, easy connection, and native integration with our frontend framework, Next.js.

We can connect to our Neon instance using the database-specific connection string using the psycopg2 library in our Python scripts (for scraping and analysis) and the pg library for our TypeScript scripts (for presentation).

User Guide & Demo

One Drive Demo Video Link (No login required)

User Guide:

- **Navigation/Dashboard overview**: The home page is a central hub containing a random stat card, a database overview, and 4 cards which you can use to navigate to:
    - Athletes: Look up individual performance histories and personal bests.
    - Schools: Explore team rosters, school records, and historical trends.
    - Meets: View results from specific track and field competitions.
    - Predictions: Access our proprietary models for upcoming championships.

    - Random Stat Card usage info:
        - Click "New Stat" to generate interesting facts
        - Click "Show SQL Query" to view the exact PostgreSQL code used to fetch that data from our Neon-hosted database.
- **Exploring athlete profiles:**
    - **Search:** enter at least 2 characters of an athlete's first or last name in the athlete lookup bar.
    - **Profile view:** Select an athlete to view their:
        - **Personal Bests:** All-time top marks categorized by Indoor/Outdoor seasons.
        - **Season Bests:** The best performance achieved in each specific year of competition.
        - **Performance History:** A chronological log of every race, including meet name, result, and wind gauge data (if applicable).
- **School and team analysis:**
    - **School Selection:** Click on any of the 10 Centennial Conference schools
    - **Roster Filters:** Use the toggle switches to filter the roster by Gender (Men/Women), Season (Indoor/Outdoor) and select the year (2010-2026).

    - **Historical Records:** Switch between the "Roster," "Top Performances," and "Season Bests" tabs to see how current athletes rank against historical team data!
- **Meet Results**
    - **Browse Meets:** Search for specific meets by name or filter by season (e.g., "2026 Indoor").
    - **Event Breakdown:** Click a meet to see a list of contested events (e.g., 60m, Mile, Shot Put).
    - **Detailed Scoring:** Select an event to view the full leaderboard, including place, athlete name, school, and result.
- **Championship Predictions:** allows users to forecast team and individual results based on three distinct models:
    - **Season Best:** Predicts scores based on the top marks achieved in the current season.
    - **Linear Regression:** Projects results based on performance trajectories over time.
    - **Average Season Performance:** Uses the mean of an athlete's performances to estimate a stable expectation.
    - **How to use:**
        1. Select the **Season** and **Gender**.
        2. Choose a **Prediction Model** from the sidebar.
        3. View the **Team Projected Scores** (leaderboard) on the right and the individual event podium projections in the center

Data Collection

Our data scraping code is in the scraping_tffrs subdirectory in our repository. The most important files are scrape.py and repository.py. The other files are tests, or secondary runs of edited code to account for bugs and other errors.

Our strategy for collecting data was to use the TFRRS page that had all of a school's performances for a season. By analysing patterns in the query parameters for each page, we were able to get the correct URL for all schools and seasons of interest, and use the URLs to download the raw HTML for each of these pages. We utilized the BeautifulSoup Python package to scrape the HTML of each page. The data we were looking for was scattered in links, lists, and raw text, but it was all extractable with BeautifulSoup. We had to adjust our database schema slightly because some data we were hoping to get was not easily available.

While investigating the page layout, we realized that we could use TFFRS's generated primary keys for tables like athletes, performances, events, etc. In many results, there was a hyperlink to the performances page. We realized this hyperlink contains a

machine key that uniquely identified athletes, performances, events, etc. By using these machine keys instead of our own, we knew all primary and foreign keys while scraping, which made inserting into the database while avoiding duplicates and ensuring referential integrity trivially easy.

We had to manually add data, but we focused on minimizing this amount. We settled on manually adding school information, location information, and identifying which events were counted towards the championships. All of this is included in the db_generating/add_manual_info.sql.

There is no user-input data. All data is collected as TFFRS, which we treat as our single source of truth.

Code Execution

Scraping Data

To run our scraping code inside the scrape_tffrs subfolder, you can run our scrape.py like a normal python script with `python scrape.py`. A .env file with the database connection string is necessary to upload our data to the database, so it can be accessed by the website. To change the schools and the seasons that the scraper will look at, edit the SCHOOLS and SEASONS variables within the download_page.py file.

Website

It isn't possible to run our Next.js website locally without the necessary environment secrets; however, our website is fully up to date on our deployment (see link on front page).

On the website, users can click on the athlete, school, and meet boxes to view information about the respective entities. Our pages are well-connected, that is, almost everything acts as a hyperlink. Athletes listed on a school page are links to the athlete's own page, and performances on an athlete's page link to the meet of the performance. Additionally, on the home page, there is a 'random stat' box. This box displays a random statistic about the data we collected. This feature was a fun way for us to explore the data while using some of the SQL queries we had previously created for Phase 1. Finally, the predictions box brings users to the prediction page. There, users can select a prediction strategy and a season, and view the predicted results of each event, along with the aggregate team scores on the right-hand side.

Analysis

We only have one analysis done inside the prediction_analysis folder. Inside, there is the linear.py file. You can just run it like a normal python script with `python linear.py`. A .env file with the database connection string is necessary to upload our predictions to the database, so it can be accessed by the website. To change which seasons the predictions are generated for, users can change the main method in linear.py.

SQL Query Examples

You can find a full list of all of our SQL Queries, as well as where they're located and what they're for, in the doc labeled SQL CODE DOCUMENTATION that can be found in the zip file. To give some context however, here are all the file paths that include SQL queries, not including the SQL queries used for building the database.

prediction_analysis\linear.py
db_generating\queries.sql
predict-the-centenni-podium\app\api\athletes\route.ts
predict-the-centenni-podium\app\api\athletes\[id]\route.ts
predict-the-centenni-podium\app\api\meets\route.ts
predict-the-centenni-podium\app\api\meets\[id]\route.ts
predict-the-centenni-podium\app\api\predictions\route.ts
predict-the-centenni-podium\app\api\randomStat\route.ts
predict-the-centenni-podium\app\api\schools\route.ts
predict-the-centenni-podium\app\api\schools\[id]\route.ts
predict-the-centenni-podium\app\api\stats\route.ts
scrape_tffrs\repository.py

Specialization

An area of major specialization was complex extraction of real data from real sources. We were scrapping data directly from TFFRS without a structured API. We had to utilize webscrapping libraries and inspect TFFRS pages manually to figure out patterns to collect the data we wanted. While this effort required many extra hours of work, it was worth completing our analysis with data about ourselves and our own teammates.

Another area of specialization was data mining for predicting centennial conference championship meet results. We created three different prediction models utilizing different calculation methods and different factors.

Another area of specialization was our GUI. We decided we wanted our project to not just be a barebores interface to a database, but an interactive and aesthetically pleasing website that can be intuitively used by other athletes and coaches alike. We built a fullstack project using the TypeScript/React/Next.js stack. This stack has the added benefit of having native deployment on Vercel and being much faster to load than TFFRS.

Strengths

We were really proud of how usable and intuitive our website is for users. We were able to present the website to some of our teammates and coaches, and they were able to gain value out of it without needing any instructions.

Our project also handles similar-but-different-data well. This means data that shares a lot of characteristics, but differs in fundamental ways. For example, relays and individual performances share attributes like meetID, resultValue, but differ because of the entity they belong to (team vs athlete). To combine these differences, we created interfaces/views that had a unified view for our website to render. We would have specialized methods for populating the view for each type of data. This allowed our code to stay modular and clean.

Limitations & Next Steps

One issue was with incorrect TFFRS data. For example, in the 2025 Outdoor season, Jonathan Cubus from Swathmore ran a 100m dash in a blistering 14 hours, 46 minutes, and 57 seconds (real stat). This result would mess up our prediction models. Our solution was to discard all predictions made with this specific result. However, in the future, we would like to implement earlier outlier detection to prevent such data points from entering our system.

Another extension for this project is to make it real-time. Our project currently requires multiple manual commands to be run to scrape new data and run our analysis on it. This limits our project's ability to stay current. New Centennial Conference track and field data comes in every weekend. Having a simple pipeline that can coordinate all data updates would be a huge step towards keeping our project up to date at all times.

We also hope to build more advanced prediction models. For the scope of this project, our current models are simple, since our focus was on building our database.

We can also extend our project to cover different conferences, not just the Centennial Conference.

Appendix

Potential English Questions

Below is a list of questions that can be answered from our database, copied from our Phase 1 submission, with slight adjustments made since we didn't collect some data that we originally planned on (meet location, meet records).

- What is the personal-best (PB) time for Mirra Klimov in the 100m Dash?
- Which athletes from Ursinus College competed in the 2025 Indoor Season?
- Who are the top 3 athletes in the Javelin Throw for the 2025 Outdoor Season from each university in the Centennial Conference?
- How many athletes from the Centennial Conference competed in the Decathlon at the 2024 National Championships?
- What is the personal best for every current athlete from a school in Baltimore in the Long Jump?
- Which meets were held on Saturday, April 12, 2025 that CC athletes attended
- Which CC athletes competed in both the 400m Dash and the 4x400m Relay at the Penn Relays in 2024?
- What are the team scores for Johns Hopkins University at the 2024 Outdoor Conference Championship for Men?
- What are the results from all races for Spencer Ye in 2024?
- What is the season record in the 200m Dash for Alex Colletti in each season?
- Who won the 800m dash in the 2024 Centenial Conference Outdoor Championships?
- Which athlete has competed in the most relays?
- How many seniors does each school have?
- What are all of Mirra Klimov's personal bests in all events she ran?
- How many times did Spencer Ye run a season best in 2025?