МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт з лабораторної роботи №1

з курсу «Аналіз даних в інформаційних системах»

на тему: «Створення сховища даних»

**Викладач:**

Олійник Ю.О.

**Виконали:**

студенти 2 курсу групи ІП-21 ФІОТ

Голованьов Г.О.

Загребельний О.А.

Київ 2024

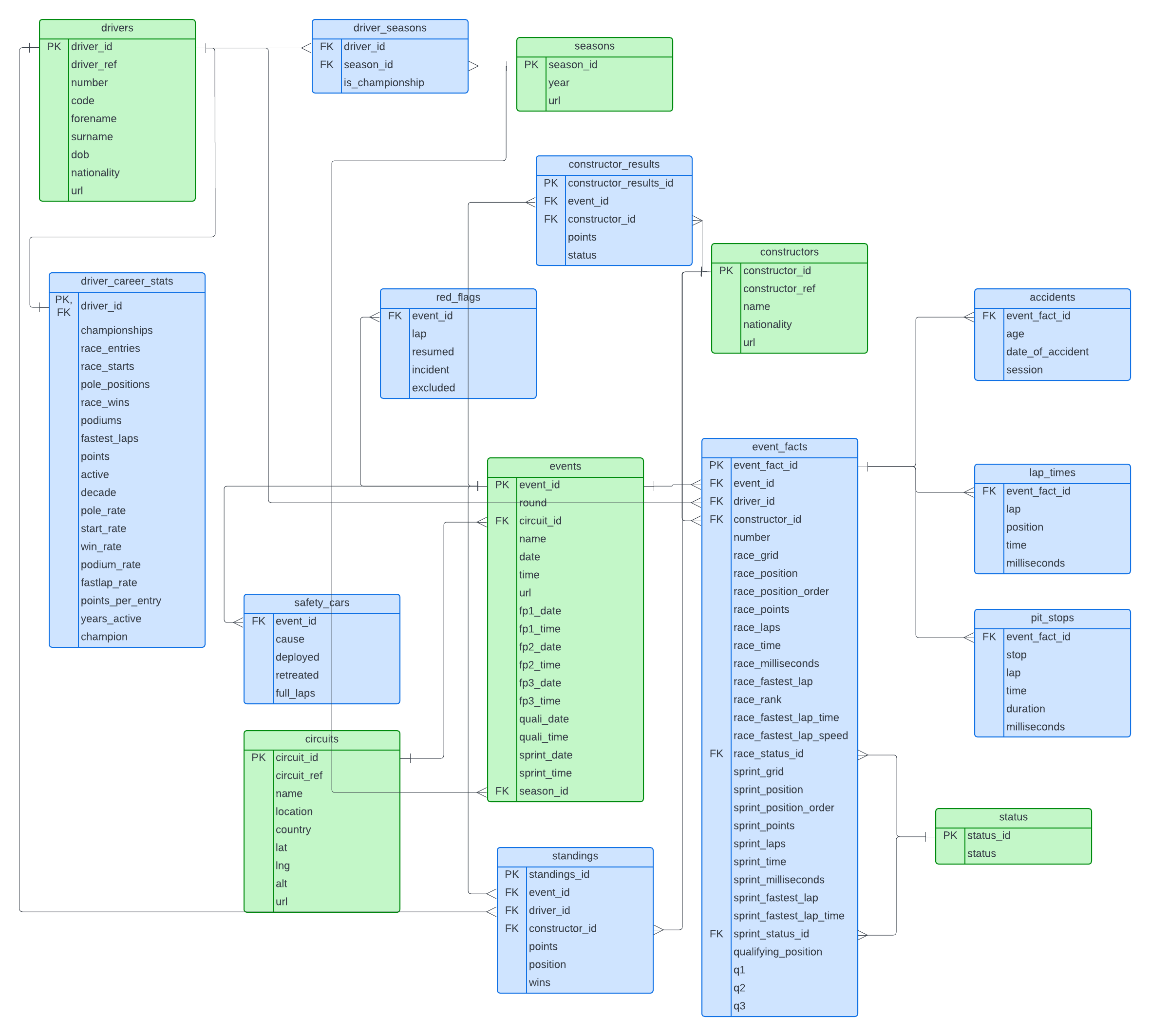
**Джерела даних**

Для виконання лабораторної роботи було обрано 3 датасети із сайту <https://www.kaggle.com/>:

1. Загальна інформація про чемпіонат Формули 1 (1950 - 2023): <https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020/data>
2. Детальна інформація про гонщиків: <https://www.kaggle.com/datasets/petalme/f1-drivers-dataset>
3. Інформація про інциденти під час перегонів: <https://www.kaggle.com/datasets/jtrotman/formula-1-race-events>

Спільними вимірами у цих джерелах є гонщики, етапи чемпіонтату, команди, сезони, статуси завершення гонки та гоночні траси.

**Таблиці вимірів та фактів**

****

Зелені таблиці – виміри;

Сині таблиці – факти;

**Модель stage зони та основного сховища**

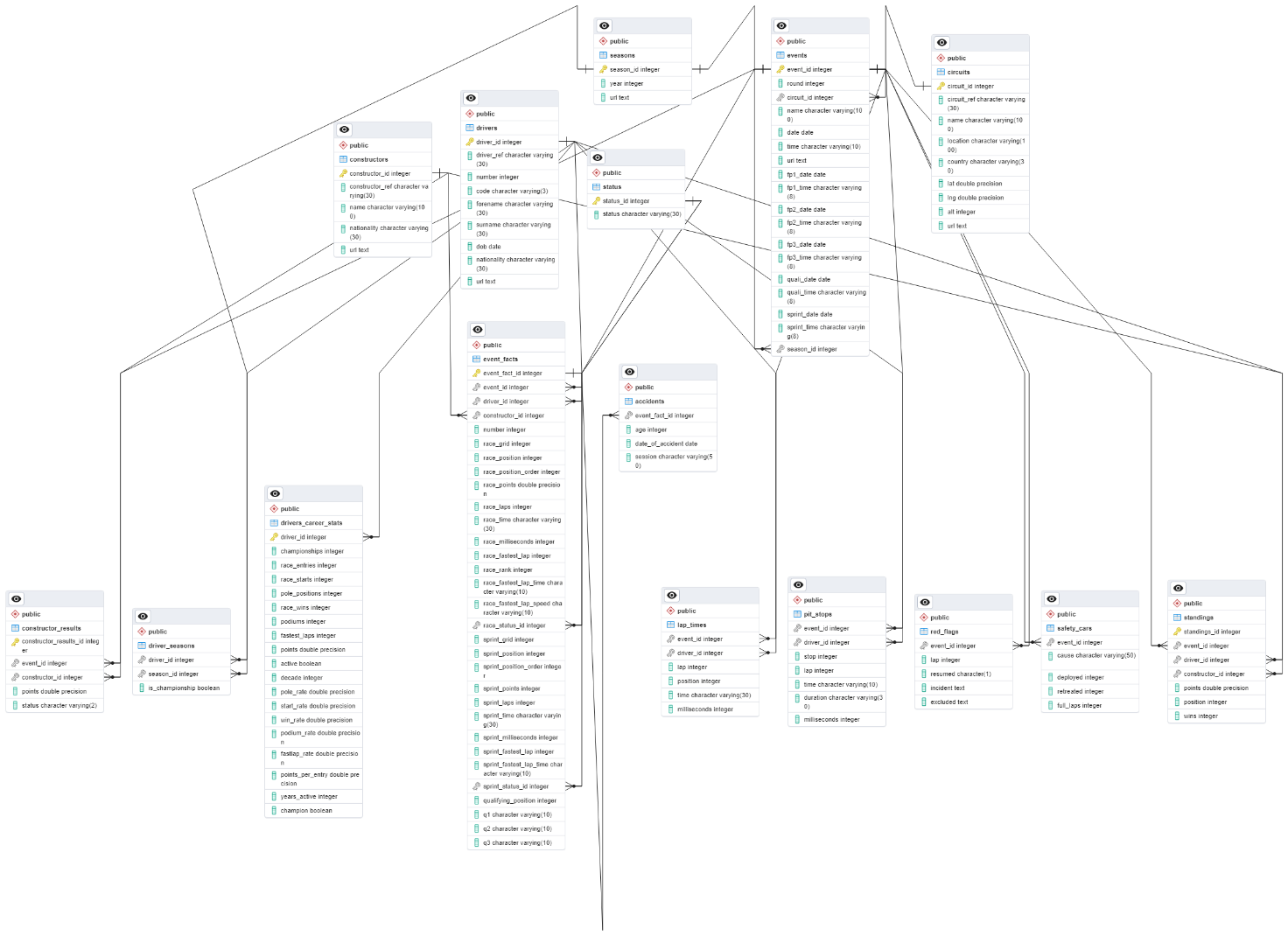
****

Рис. 1 – Cховище за типом «сніжинка»

Скрипти створення основного сховища знаходяться у додатку А.

Опис таблиць сховища:

1. **Circuits (Траси)**: Зберігає дані про формульні траси, включаючи їх назви, розташування та координати, а також іншу загальну інформацію.
2. **Constructors (Конструктори)**: Містить інформацію про команди-конструктори, включаючи їх назви та національності.
3. **Drivers (Водії)**: Зберігає дані про водіїв, такі як їх ім'я, прізвище, дату народження, національність та інші особисті дані.
4. **Drivers Career Stats (Статистика кар'єри водіїв)**: Включає статистичні дані про кар'єрні досягнення водіїв у світі Формули 1, такі як кількість чемпіонатів, кількість стартів, перемоги, підіуми, найшвидші кола тощо.
5. **Seasons (Сезони)**: Містить дані про сезони Формули 1, такі як рік і посилання на додаткову інформацію.
6. **Driver Seasons (Сезони водіїв)**: Визначає зв'язок між водіями і сезонами, вказуючи, які водії брали участь в яких сезонах.
7. **Status (Статус)**: Зберігає список можливих статусів подій, таких як "Завершено", "Не завершено", "Дискваліфіковано" і т. д.
8. **Events (Події)**: Містить дані про кожну окрему подію Формули 1, такі як траса, дата, назва, раунд, сезон, часи сесій, тощо.
9. **Constructor Results (Результати конструкторів)**: Зберігає результати кожної події для кожного конструктора, включаючи отримані очки та статус.
10. **Lap Times (Часи кола)**: Містить дані про часи кожного кола кожного водія під час кожної події.
11. **Standings (Рейтинги)**: Містить дані про позиції, очки та перемоги водіїв та конструкторів після кожної події.
12. **Pit Stops (Зупинки в боксах)**: Зберігає дані про зупинки в боксах під час кожної події, такі як номер кола, тривалість зупинки тощо.
13. **Event Facts (Факти подій)**: Містить додаткові факти про кожну подію, такі як позиції на старті, позиції в гонці, найшвидші кола тощо.
14. **Accidents (Аварії)**: Зберігає дані про будь-які аварії, які сталися під час подій, включаючи вік, дату та тип аварії.
15. **Safety Cars (Автомобілі безпеки)**: Містить інформацію про використання автомобілів безпеки під час кожної події.
16. **Red Flags (Червоні прапори)**: Зберігає дані про будь-які червоні прапори, що були підняті під час подій, включаючи довжину перерви та пов'язані події.

**ETL засоби**

Файли csv з обраних джерел було імпортовано як вхідні дані. Щоб поєднати дані з різних датасетів були використані скрипти на Python, що наведені в додатку Б, а після отримання коректних вихідних файлів було дані з них було імпортовано у сховище засобами СУБД PostgreSQL.

**Додаток А**

(Скрипти для створення сховища)

/\*---------------------------------------------------

CIRCUITS

---------------------------------------------------\*/

CREATE TABLE circuits (

    circuit\_id SERIAL PRIMARY KEY,

    circuit\_ref VARCHAR(30),

    name VARCHAR(100),

    location VARCHAR(100),

    country VARCHAR(30),

    lat FLOAT,

    lng FLOAT,

    alt INT,

    url TEXT

);

/\*---------------------------------------------------

CONSTRUCTORS

---------------------------------------------------\*/

CREATE TABLE constructors (

    constructor\_id SERIAL PRIMARY KEY,

    constructor\_ref VARCHAR(30),

    name VARCHAR(100),

    nationality VARCHAR(30),

    url TEXT

);

/\*---------------------------------------------------

DRIVERS

---------------------------------------------------\*/

CREATE TABLE drivers (

    driver\_id SERIAL PRIMARY KEY,

    driver\_ref VARCHAR(30),

    number INT,

    code VARCHAR(3),

    forename VARCHAR(30),

    surname VARCHAR(30),

    dob DATE,

    nationality VARCHAR(30),

    url TEXT

);

/\*---------------------------------------------------

DRIVERS CAREER STATS

---------------------------------------------------\*/

CREATE TABLE drivers\_career\_stats (

    driver\_id INT PRIMARY KEY REFERENCES drivers(driver\_id),

    championships INT,

    race\_entries INT,

    race\_starts INT,

    pole\_positions INT,

    race\_wins INT,

    podiums INT,

    fastest\_laps INT,

    points FLOAT,

    active BOOLEAN,

    decade INT,

    pole\_rate FLOAT,

    start\_rate FLOAT,

    win\_rate FLOAT,

    podium\_rate FLOAT,

    fastlap\_rate FLOAT,

    points\_per\_entry FLOAT,

    years\_active INT,

    champion BOOLEAN

);

/\*---------------------------------------------------

SEASONS

---------------------------------------------------\*/

CREATE TABLE seasons (

    season\_id SERIAL PRIMARY KEY,

    year INT,

    url TEXT

);

/\*---------------------------------------------------

DRIVER SEASONS

---------------------------------------------------\*/

CREATE TABLE driver\_seasons (

    driver\_id INT REFERENCES drivers(driver\_id),

    season\_id INT REFERENCES seasons(season\_id),

    is\_championship BOOLEAN

);

/\*---------------------------------------------------

STATUS

---------------------------------------------------\*/

CREATE TABLE status (

    status\_id SERIAL PRIMARY KEY,

    status VARCHAR(30)

);

/\*---------------------------------------------------

EVENTS

---------------------------------------------------\*/

CREATE TABLE events (

    event\_id SERIAL PRIMARY KEY,

    round INT,

    circuit\_id INT REFERENCES circuits(circuit\_id),

    name VARCHAR(100),

    date DATE,

    time VARCHAR(10),

    url TEXT,

    fp1\_date date,

    fp1\_time VARCHAR(8),

    fp2\_date date,

    fp2\_time VARCHAR(8),

    fp3\_date date,

    fp3\_time VARCHAR(8),

    quali\_date date,

    quali\_time VARCHAR(8),

    sprint\_date date,

    sprint\_time VARCHAR(8),

    season\_id INT REFERENCES seasons(season\_id)

);

/\*---------------------------------------------------

CONSTRUCTOR RESULTS

---------------------------------------------------\*/

CREATE TABLE constructor\_results (

    constructor\_results\_id SERIAL PRIMARY KEY,

    event\_id INT REFERENCES events(event\_id),

    constructor\_id INT REFERENCES constructors(constructor\_id),

    points FLOAT,

    status VARCHAR(2)

);

/\*---------------------------------------------------

LAP TIMES

---------------------------------------------------\*/

CREATE TABLE lap\_times (

    event\_id INT REFERENCES events(event\_id),

    driver\_id INT REFERENCES drivers(driver\_id),

    lap INT,

    position INT,

    time VARCHAR(30),

    milliseconds INT

);

/\*---------------------------------------------------

STANDINGS

---------------------------------------------------\*/

CREATE TABLE standings (

    standings\_id SERIAL PRIMARY KEY,

    event\_id INT REFERENCES events(event\_id),

    driver\_id INT REFERENCES drivers(driver\_id),

    constructor\_id INT REFERENCES constructors(constructor\_id),

    points FLOAT,

    position INT,

    wins INT

);

/\*---------------------------------------------------

PIT STOPS

---------------------------------------------------\*/

CREATE TABLE pit\_stops (

    event\_id INT REFERENCES events(event\_id),

    driver\_id INT REFERENCES drivers(driver\_id),

    stop INT,

    lap INT,

    time VARCHAR(10),

    duration VARCHAR(30),

    milliseconds INT

);

/\*---------------------------------------------------

EVENT FACTS

---------------------------------------------------\*/

CREATE TABLE event\_facts (

    event\_fact\_id SERIAL PRIMARY KEY,

    event\_id INT REFERENCES events(event\_id),

    driver\_id INT REFERENCES drivers(driver\_id),

    constructor\_id INT REFERENCES constructors(constructor\_id),

    number INT,

    race\_grid INT,

    race\_position INT,

    race\_position\_order INT,

    race\_points FLOAT,

    race\_laps INT,

    race\_time VARCHAR(30),

    race\_milliseconds INT,

    race\_fastest\_lap INT,

    race\_rank INT,

    race\_fastest\_lap\_time VARCHAR(10),

    race\_fastest\_lap\_speed VARCHAR(10),

    race\_status\_id INT REFERENCES status(status\_id),

    sprint\_grid INT,

    sprint\_position INT,

    sprint\_position\_order INT,

    sprint\_points INT,

    sprint\_laps INT,

    sprint\_time VARCHAR(30),

    sprint\_milliseconds INT,

    sprint\_fastest\_lap INT,

    sprint\_fastest\_lap\_time VARCHAR(10),

    sprint\_status\_id INT REFERENCES status(status\_id),

    qualifying\_position INT,

    q1 VARCHAR(10),

    q2 VARCHAR(10),

    q3 VARCHAR(10)

);

/\*---------------------------------------------------

ACCIDENTS

---------------------------------------------------\*/

CREATE TABLE accidents (

    event\_fact\_id INT REFERENCES event\_facts(event\_fact\_id),

    age INT,

    date\_of\_accident DATE,

    session VARCHAR(50)

);

/\*---------------------------------------------------

SAFETY CARS

---------------------------------------------------\*/

CREATE TABLE safety\_cars (

    event\_id INT REFERENCES events(event\_id),

    cause VARCHAR(50),

    deployed INT,

    retreated INT,

    full\_laps INT

);

/\*---------------------------------------------------

RED FLAGS

---------------------------------------------------\*/

CREATE TABLE red\_flags (

    event\_id INT REFERENCES events(event\_id),

    lap INT,

    resumed CHAR(1),

    incident TEXT,

    excluded TEXT

);

**Додаток Б**

(Скрипти Python)

accidents.py

import pandas as pd

import numpy as np

from datetime import datetime, timedelta

from config import INPUT\_CSV\_DIR, OUTPUT\_CSV\_DIR, join

accidents\_df = pd.read\_csv(join(INPUT\_CSV\_DIR, 'fatal\_accidents\_drivers.csv'))

drivers\_df = pd.read\_csv(join(INPUT\_CSV\_DIR, 'drivers.csv'))

races\_df = pd.read\_csv(join(INPUT\_CSV\_DIR, 'races.csv'))

event\_facts\_df = pd.read\_csv(join(OUTPUT\_CSV\_DIR, 'event\_facts.csv'))

def concat\_name(row):

    name\_parts = row['Driver'].split()

    surname = name\_parts[-1]

    forename = ' '.join(name\_parts[:-1])

    return pd.Series([forename, surname])

def concat\_race(row):

    race = str(row['year']) + ' ' + row['name']

    return pd.Series([race])

accidents\_df[['forename', 'surname']] = accidents\_df.apply(concat\_name, axis=1)

races\_df['Event'] = races\_df.apply(concat\_race, axis=1)

merged\_df = pd.merge(drivers\_df[['driverId', 'forename', 'surname']], accidents\_df, on=['forename', 'surname'], how='inner')

merged\_df = pd.merge(merged\_df, races\_df[['raceId', 'Event']], on=['Event'], how='inner')

merged\_df = pd.merge(merged\_df, event\_facts\_df[['resultId', 'eventId', 'driverId', 'constructorId']], left\_on=['raceId', 'driverId'], right\_on=['eventId', 'driverId'], how='inner')

merged\_df.drop(['forename', 'surname', 'Driver', 'Event'], axis=1, inplace=True)

merged\_df.rename(columns={'resultId': 'eventFactId'}, inplace=True)

merged\_df['Date Of Accident'] = pd.to\_datetime(merged\_df['Date Of Accident'], format='%m/%d/%y')

future\_dates = merged\_df['Date Of Accident'] > datetime.now()

merged\_df.loc[future\_dates, 'Date Of Accident'] -= timedelta(days=36525)

merged\_df['Age'].replace('', np.nan, inplace=True)

merged\_df['Age'] = merged\_df['Age'].astype(float).astype('Int64')

merged\_df = merged\_df[['eventFactId', 'Age', 'Date Of Accident', 'Session']]

merged\_df.to\_csv(join(OUTPUT\_CSV\_DIR, 'accidents.csv'), index=False)

config.py

import os

def join(\*args):

    return os.path.join(\*args)

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

INPUT\_CSV\_DIR = os.path.join(BASE\_DIR, 'input\_csv')

OUTPUT\_CSV\_DIR = os.path.join(BASE\_DIR, 'output\_csv')

drivers.py

import pandas as pd

import ast

from config import INPUT\_CSV\_DIR, OUTPUT\_CSV\_DIR, join

drivers\_path = join(INPUT\_CSV\_DIR, 'drivers.csv')

f1\_drivers\_path = join(INPUT\_CSV\_DIR, 'F1Drivers\_Dataset.csv')

seasons\_path = join(OUTPUT\_CSV\_DIR, 'seasons.csv')

drivers = pd.read\_csv(drivers\_path)

f1\_drivers = pd.read\_csv(f1\_drivers\_path)

drivers['full\_name'] = drivers['forename'] + ' ' + drivers['surname']

merged\_df = pd.merge(drivers, f1\_drivers, left\_on='full\_name', right\_on='Driver', how='left')

merged\_df = merged\_df.drop('full\_name', axis=1)

output\_drivers\_cols = ['driverId', 'driverRef', 'number', 'code', 'forename', 'surname', 'dob', 'nationality', 'url']

output\_drivers = merged\_df[output\_drivers\_cols]

output\_drivers.to\_csv(join(OUTPUT\_CSV\_DIR, 'drivers.csv'), index=False)

career\_stats\_cols = ['driverId', 'Championships', 'Race\_Entries', 'Race\_Starts', 'Pole\_Positions', 'Race\_Wins', 'Podiums', 'Fastest\_Laps', 'Points', 'Active', 'Decade', 'Pole\_Rate', 'Start\_Rate', 'Win\_Rate', 'Podium\_Rate', 'FastLap\_Rate', 'Points\_Per\_Entry', 'Years\_Active', 'Champion']

career\_stats = merged\_df[career\_stats\_cols]

career\_stats.to\_csv(join(OUTPUT\_CSV\_DIR, 'driver\_career\_stats.csv'), index=False)

seasons = pd.read\_csv(seasons\_path)

driver\_seasons = pd.DataFrame(columns=['driverId', 'seasonId', 'isChampionship'])

for index, row in merged\_df.iterrows():

    driver\_id = row['driverId']

    seasons\_list = ast.literal\_eval(row['Seasons'])  # convert string to list

    championship\_years = ast.literal\_eval(row['Championship Years']) if isinstance(row['Championship Years'], str) else []  # convert string to list

    for season\_year in seasons\_list:

        season\_id = seasons.loc[seasons['year'] == season\_year, 'seasonId'].values[0]

        is\_championship = season\_year in championship\_years

        driver\_seasons = driver\_seasons.\_append({'driverId': driver\_id, 'seasonId': season\_id, 'isChampionship': is\_championship}, ignore\_index=True)

driver\_seasons.to\_csv(join(OUTPUT\_CSV\_DIR, 'driver\_seasons.csv'), index=False)

event\_facts.py

import pandas as pd

import numpy as np

from config import INPUT\_CSV\_DIR, OUTPUT\_CSV\_DIR, join

results\_path = join(INPUT\_CSV\_DIR, 'results.csv')

sprint\_results\_path = join(INPUT\_CSV\_DIR, 'sprint\_results.csv')

qualifying\_path = join(INPUT\_CSV\_DIR, 'qualifying.csv')

results\_df = pd.read\_csv(results\_path)

sprint\_results\_df = pd.read\_csv(sprint\_results\_path)

qualifying\_df = pd.read\_csv(qualifying\_path)

results\_df.replace({'\\N': np.nan}, inplace=True)

sprint\_results\_df.replace({'\\N': np.nan}, inplace=True)

qualifying\_df.replace({'\\N': np.nan}, inplace=True)

results\_df['number'] = results\_df['number'].astype('Int64')

merged\_df = pd.merge(results\_df, sprint\_results\_df, on=['raceId', 'driverId', 'constructorId', 'number'], how='outer', suffixes=('\_race', '\_sprint'))

event\_facts\_df = pd.merge(merged\_df, qualifying\_df, on=['raceId', 'driverId', 'constructorId', 'number'], how='left')

event\_facts\_df = event\_facts\_df.rename(columns={

    'resultId\_race': 'resultId',

    'raceId': 'eventId',

    'grid\_race': 'race\_grid',

    'position\_race': 'race\_position',

    'positionText\_race': 'race\_positionText',

    'positionOrder\_race': 'race\_positionOrder',

    'points\_race': 'race\_points',

    'laps\_race': 'race\_laps',

    'time\_race': 'race\_time',

    'milliseconds\_race': 'race\_milliseconds',

    'fastestLap\_race': 'race\_fastestLap',

    'rank': 'race\_rank',

    'fastestLapTime\_race': 'race\_fastestLapTime',

    'fastestLapSpeed': 'race\_fastestLapSpeed',

    'statusId\_race': 'race\_statusId',

    'resultId\_sprint': 'sprintResultId',

    'grid\_sprint': 'sprint\_grid',

    'position\_sprint': 'sprint\_position',

    'positionText\_sprint': 'sprint\_positionText',

    'positionOrder\_sprint': 'sprint\_positionOrder',

    'points\_sprint': 'sprint\_points',

    'laps\_sprint': 'sprint\_laps',

    'time\_sprint': 'sprint\_time',

    'milliseconds\_sprint': 'sprint\_milliseconds',

    'fastestLap\_sprint': 'sprint\_fastestLap',

    'fastestLapTime\_sprint': 'sprint\_fastestLapTime',

    'statusId\_sprint': 'sprint\_statusId',

    'position': 'qualifying\_position'

})

event\_facts\_df = event\_facts\_df[[

    'resultId', 'eventId', 'driverId', 'constructorId', 'number',

    'race\_grid', 'race\_position', 'race\_positionText', 'race\_positionOrder', 'race\_points',

    'race\_laps', 'race\_time', 'race\_milliseconds', 'race\_fastestLap', 'race\_rank',

    'race\_fastestLapTime', 'race\_fastestLapSpeed', 'race\_statusId',

    'sprint\_grid', 'sprint\_position', 'sprint\_positionText', 'sprint\_positionOrder', 'sprint\_points',

    'sprint\_laps', 'sprint\_time', 'sprint\_milliseconds', 'sprint\_fastestLap', 'sprint\_fastestLapTime', 'sprint\_statusId',

    'qualifying\_position', 'q1', 'q2', 'q3'

]]

event\_facts\_df = event\_facts\_df.drop(['race\_positionText', 'sprint\_positionText'], axis=1)

event\_facts\_df.to\_csv(join(OUTPUT\_CSV\_DIR, 'event\_facts.csv'), index=False)

events.py

import pandas as pd

from config import INPUT\_CSV\_DIR, OUTPUT\_CSV\_DIR, join

races\_path = join(INPUT\_CSV\_DIR, 'races.csv')

seasons\_path = join(OUTPUT\_CSV\_DIR, 'seasons.csv')

races = pd.read\_csv(races\_path)

seasons = pd.read\_csv(seasons\_path)

year\_season\_map = dict(zip(seasons['year'], seasons['seasonId']))

races['seasonId'] = races['year'].map(year\_season\_map)

events = races.drop('year', axis=1)

events = events.rename(columns={'raceId': 'eventId'})

events.to\_csv(join(OUTPUT\_CSV\_DIR, 'events.csv'), index=False)

red\_flags.py

import pandas as pd

from config import INPUT\_CSV\_DIR, OUTPUT\_CSV\_DIR, join

red\_flags\_path = join(INPUT\_CSV\_DIR, 'red\_flags.csv')

races\_path = join(INPUT\_CSV\_DIR, 'races.csv')

red\_flags\_df = pd.read\_csv(red\_flags\_path)

races\_df = pd.read\_csv(races\_path)

def concat\_race(row):

    race = str(row['year']) + ' ' + row['name']

    return pd.Series([race])

races\_df['Race'] = races\_df.apply(concat\_race, axis=1)

merged\_df = pd.merge(races\_df[['raceId', 'Race']], red\_flags\_df, on=['Race'], how='inner')

merged\_df.drop(['Race'], axis=1, inplace=True)

merged\_df = merged\_df.rename(columns={'raceId': 'eventId'})

merged\_df.to\_csv(join(OUTPUT\_CSV\_DIR, 'red\_flags.csv'), index=False)

safety\_cars.py

import pandas as pd

import numpy as np

from config import INPUT\_CSV\_DIR, OUTPUT\_CSV\_DIR, join

safety\_cars\_path = join(INPUT\_CSV\_DIR, 'safety\_cars.csv')

races\_path = join(INPUT\_CSV\_DIR, 'races.csv')

safety\_cars\_df = pd.read\_csv(safety\_cars\_path)

races\_df = pd.read\_csv(races\_path)

def concat\_race(row):

    race = str(row['year']) + ' ' + row['name']

    return pd.Series([race])

races\_df['Race'] = races\_df.apply(concat\_race, axis=1)

merged\_df = pd.merge(races\_df[['raceId', 'Race']], safety\_cars\_df, on=['Race'], how='inner')

merged\_df.drop(['Race'], axis=1, inplace=True)

merged\_df['Retreated'].replace('', np.nan, inplace=True)

merged\_df['Retreated'] = merged\_df['Retreated'].astype(float).astype('Int64')

merged\_df = merged\_df.rename(columns={'raceId': 'eventId'})

merged\_df.to\_csv(join(OUTPUT\_CSV\_DIR, 'safety\_cars.csv'), index=False)

seasons.py

import pandas as pd

import numpy as np

from config import INPUT\_CSV\_DIR, OUTPUT\_CSV\_DIR, join

safety\_cars\_path = join(INPUT\_CSV\_DIR, 'safety\_cars.csv')

races\_path = join(INPUT\_CSV\_DIR, 'races.csv')

safety\_cars\_df = pd.read\_csv(safety\_cars\_path)

races\_df = pd.read\_csv(races\_path)

def concat\_race(row):

    race = str(row['year']) + ' ' + row['name']

    return pd.Series([race])

races\_df['Race'] = races\_df.apply(concat\_race, axis=1)

merged\_df = pd.merge(races\_df[['raceId', 'Race']], safety\_cars\_df, on=['Race'], how='inner')

merged\_df.drop(['Race'], axis=1, inplace=True)

merged\_df['Retreated'].replace('', np.nan, inplace=True)

merged\_df['Retreated'] = merged\_df['Retreated'].astype(float).astype('Int64')

merged\_df = merged\_df.rename(columns={'raceId': 'eventId'})

merged\_df.to\_csv(join(OUTPUT\_CSV\_DIR, 'safety\_cars.csv'), index=False)

standings.py

import pandas as pd

import numpy as np

from config import INPUT\_CSV\_DIR, OUTPUT\_CSV\_DIR, join

driver\_standings\_path = join(INPUT\_CSV\_DIR, 'driver\_standings.csv')

constructor\_standings\_path = join(INPUT\_CSV\_DIR, 'constructor\_standings.csv')

driver\_standings\_df = pd.read\_csv(driver\_standings\_path)

constructor\_standings\_df = pd.read\_csv(constructor\_standings\_path)

driver\_standings\_df['constructorId'] = np.nan

constructor\_standings\_df['driverId'] = np.nan

standings\_df = pd.concat([driver\_standings\_df, constructor\_standings\_df], ignore\_index=True)

standings\_df.reset\_index(drop=True, inplace=True)

standings\_df['standingsId'] = standings\_df.index + 1

standings\_df.rename(columns={'raceId': 'eventId'}, inplace=True)

standings\_df['driverId'] = standings\_df['driverId'].astype('Int64')

standings\_df['constructorId'] = standings\_df['constructorId'].astype('Int64')

standings\_df.drop(columns=['positionText'], inplace=True)

standings\_df = standings\_df[['standingsId', 'eventId', 'driverId', 'constructorId', 'points', 'position', 'wins']]

standings\_df.to\_csv(join(OUTPUT\_CSV\_DIR, 'standings.csv'), index=False)

Скрипт для додавання нового гонщика в сховище:

*from* config *import* env

*import* psycopg2

*import* pandas *as* pd

def driver\_exists(*cursor*, *driver*):

*cursor*.execute("SELECT COUNT(\*) FROM drivers WHERE forename = %s AND surname = %s", (*driver*['forename'], *driver*['surname']))

    count = *cursor*.fetchone()[0]

*return* count > 0

def insert\_driver(*cursor*, *driver*):

*cursor*.execute("INSERT INTO drivers (driver\_ref, number, code, forename, surname, dob, nationality, url) \

                   VALUES (%s, %s, %s, %s, %s, %s, %s, %s) RETURNING driver\_id",

                   (*driver*['driverRef'],

*driver*['number'],

*driver*['code'],

*driver*['forename'],

*driver*['surname'],

*driver*['dob'],

*driver*['nationality'],

*driver*['url']))

    inserted\_id = *cursor*.fetchone()[0]

*cursor*.execute("INSERT INTO drivers\_career\_stats (driver\_id, championships, race\_entries, race\_starts, pole\_positions, \

                    race\_wins, podiums, fastest\_laps, points, active, decade, pole\_rate, start\_rate, win\_rate, \

                    podium\_rate, fastLap\_rate, points\_per\_entry, years\_active, champion) \

                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)",

                   (inserted\_id,

*driver*['Championships'],

*driver*['Race\_Entries'],

*driver*['Race\_Starts'],

*driver*['Pole\_Positions'],

*driver*['Race\_Wins'],

*driver*['Podiums'],

*driver*['Fastest\_Laps'],

*driver*['Points'],

*driver*['Active'],

*driver*['Decade'],

*driver*['Pole\_Rate'],

*driver*['Start\_Rate'],

*driver*['Win\_Rate'],

*driver*['Podium\_Rate'],

*driver*['FastLap\_Rate'],

*driver*['Points\_Per\_Entry'],

*driver*['Years\_Active'],

*driver*['Champion']))

*return* inserted\_id

def main():

    drivers\_df = pd.read\_csv('input\_csv/new\_drivers.csv')

    connection = psycopg2.connect(

*database*=env.DB\_NAME,

*user*=env.DB\_USER,

*password*=env.DB\_PASSWORD,

*host*=env.DB\_HOST,

*port*=env.DB\_PORT,

*client\_encoding*=env.CLIENT\_ENCODING

    )

    cursor = connection.cursor()

*try*:

*for* \_, driver *in* drivers\_df.iterrows():

*if* not driver\_exists(cursor, driver):

                driver\_id = insert\_driver(cursor, driver)

                print(f"Added driver {driver['forename']} {driver['surname']} to the database with ID {driver\_id}")

        connection.commit()

        print('Transaction committed succesfully.')

*except* Exception *as* e:

        connection.rollback()

        print(f'Error occured: {e}')

*finally*:

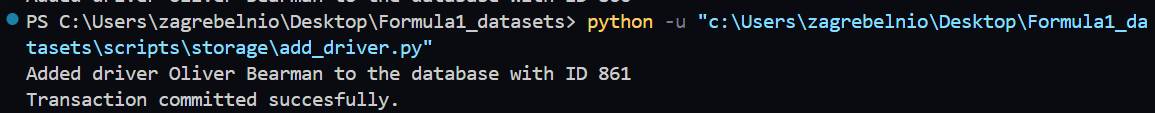
        cursor.close()

        connection.close()

*if* \_\_name\_\_ == "\_\_main\_\_":

    main()

Результат роботи скрипту:



Зображення, що містить знімок екрана, текст

Автоматично згенерований опис