**Formula 1 World Championship Analysis Report**

**Team Members and Responsibilities**

* Apoorva Chowdary Mandalapu (Responsible for ER Modelling, Schema generation (DDL/DML queries))
* Laxminarayana Vasa (Responsible for conducting Experiments and Final Report)
* Rahul Varma Alluri (Responsible for SQL Queries and Indexing)
* Vikas Vulugonipalli (Responsible for Data Preparation and Performance Analysis)

**Project Goal:**

This venture aims to scrutinize the Formula One (F1) dataset spanning 70 years. Our primary aim is to uncover trends and patterns behind drivers' failure to complete races, establish correlations between their performances during qualifying rounds and final races, and evaluate the effect of maintenance problems on their performance. To accomplish this, we will develop a sophisticated data model that will facilitate data analysis and the identification of interconnections among various variables.

This project will allow us to test the following concepts discussed in class:

* ER-Diagram and Modelling
* Transform conceptual design into schema using DDL/DML
* Bulk processing of the data
* Data retrieval from the Database (DDL/DML) based on the requirement
* Performance tuning
* Query plan interpretation and add indexes to improve
* Measure query performance before and after adding indexes

**Attached Files:**

* Modified data from source( data.zip)
* Schema file ( F1\_schema\_DDL\_DML.sql ) (without indexing)
* SQL query file( F1\_queries.sql)
* ER diagram(ER Diagram.html)
* Indexing file(F1\_project\_index\_creation)

**Dataset**

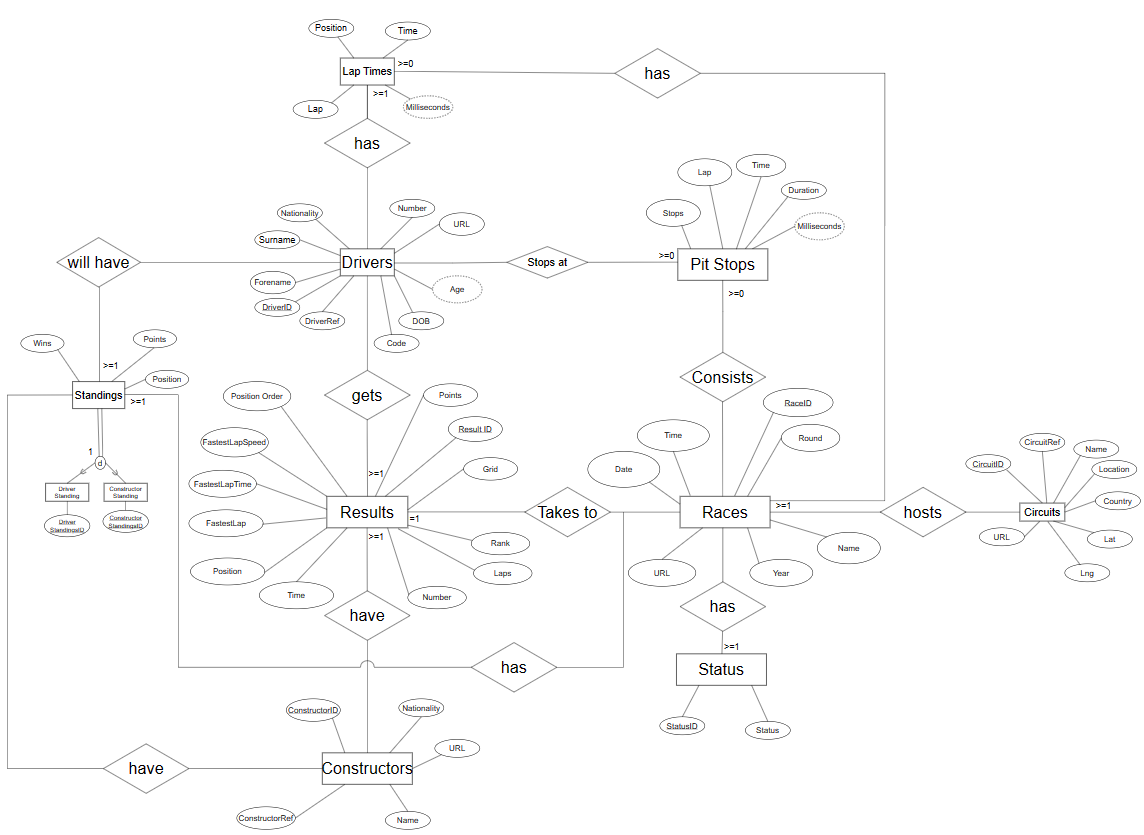
The data set was hosted on the Kaggle [1]. The dataset consists of data from sports from 1950 to 2020. The data set contains 14 different CSV files with information on drivers, circuits, lap times, constructors' results, constructor standings, races, status, pit stops, and seasons.

**Data Normalization:**

As a part of the normalization process, we have carefully reviewed and refined the original data files. Specifically, we have removed unnecessary tables and columns from certain entities to optimize the data for analysis. We aimed to ensure that only the most relevant information is retained, enabling us to conduct accurate and insightful analysis.

**Conceptual Design**

The ER diagram below conceptually represents the dataset; it lists the entities included in the data, their identifying attributes, and the relationship between them.



**Database Schema**

The following DDL statements implement the schema expressed in the ER diagram in

SQL

CREATE DATABASE F1\_project

Create or replace Function get\_age(dob DATE)

returns INTERVAL

LANGUAGE plpgsql IMMUTABLE

AS $$

BEGIN

    return age(dob);

END

$$;

CREATE OR REPLACE FUNCTION compute\_millisecond(duration DOUBLE PRECISION)

RETURNS INT

LANGUAGE plpgsql IMMUTABLE

AS $$

BEGIN

    return duration \* 1000;

END

$$;

DROP TABLE IF EXISTS public.DRIVERS CASCADE;

CREATE TABLE public.DRIVERS(

    driverid SERIAL PRIMARY KEY,

    driverRef VARCHAR(20) NOT NULL DEFAULT 'default',

    "number" INT,

    code VARCHAR(3),

    forename VARCHAR(100),

    surname VARCHAR(100),

    dob DATE NOT NULL DEFAULT '1900-01-01'::DATE,

    age INTERVAL GENERATED ALWAYS AS (get\_age(dob)) STORED,

    nationality VARCHAR(20),

    url TEXT

);

DROP TABLE IF EXISTS public.CONSTRUCTORS CASCADE;

CREATE TABLE public.CONSTRUCTORS(

    constructorId SERIAL PRIMARY KEY,

    constructorRef VARCHAR(20) NOT NULL DEFAULT 'default',

    "name" VARCHAR(30),

    nationality VARCHAR(20),

    url TEXT

);

DROP TABLE IF EXISTS public.CIRCUITS CASCADE;

CREATE TABLE public.CIRCUITS(

    circuitId SERIAL PRIMARY KEY,

    circuitRef VARCHAR(20) NOT NULL DEFAULT 'default',

    "name" VARCHAR(50),

    "location" VARCHAR(30),

    country VARCHAR (20),

    lat DOUBLE PRECISION,

    lng DOUBLE PRECISION,

    url TEXT

);

DROP TABLE IF EXISTS public.SEASONS CASCADE;

CREATE TABLE public.SEASONS(

    "year" INT PRIMARY KEY,

    url TEXT

);

DROP TABLE IF EXISTS public.STATUS CASCADE;

CREATE TABLE public.STATUS(

    statusId SERIAL PRIMARY KEY,

    status VARCHAR(30)

);

DROP TABLE IF EXISTS public.RACES CASCADE;

CREATE TABLE public.RACES(

    raceId SERIAL PRIMARY KEY,

    "year" INT,

    round INT,

    circuitId INT REFERENCES public.CIRCUITS(circuitId) NOT NULL,

    "name" VARCHAR(40),

    "date" DATE NOT NULL DEFAULT '1900-01-01'::DATE,

    "time" TIME,

    url TEXT

);

DROP TABLE IF EXISTS public.PITSTOPS CASCADE;

CREATE TABLE public.PITSTOPS(

    raceId INT REFERENCES public.RACES(raceId) NOT NULL,

    driverId INT REFERENCES public.DRIVERS(driverId) NOT NULL,

    stop INT,

    lap INT NOT NULL,

    "time" TIME,

    duration DOUBLE PRECISION,

    milliseconds INT GENERATED ALWAYS AS (compute\_millisecond(duration)) STORED, -- can be derived column

    PRIMARY KEY(raceId, driverId, lap)

);

DROP TABLE IF EXISTS public.LAPTIMES CASCADE;

CREATE TABLE public.LAPTIMES(

    raceId INT REFERENCES public.RACES(raceId) NOT NULL,

    driverId INT REFERENCES public.DRIVERS(driverId) NOT NULL,

    lap INT NOT NULL,

    "position" INT,

    "time" TIME,

    --milliseconds INT GENERATED ALWAYS AS (compute\_millisecond("time")) STORED, -- can be derived column

    PRIMARY KEY(raceId, driverId, lap)

);

DROP TABLE IF EXISTS public.RESULTS CASCADE;

CREATE TABLE public.RESULTS(

    resultId SERIAL PRIMARY KEY,

    raceId INT REFERENCES public.RACES(raceId) NOT NULL,

    driverId INT REFERENCES public.DRIVERS(driverId) NOT NULL,

    constructorId INT REFERENCES public.CONSTRUCTORS(constructorId) NOT NULL,

    "number" INT,

    grid INT,

    "position" INT,

    positionOrder INT,

    points FLOAT,

    laps INT,

    "time" INTERVAL,

    fastestLap INT,

    "rank" INT,

    fastestLapTime TIME,

    fastestLapSpeed DOUBLE PRECISION,

    statusId INT REFERENCES public.STATUS(statusId) NOT NULL

);

DROP TABLE IF EXISTS public.DRIVER\_STANDINGS CASCADE;

CREATE TABLE public.DRIVER\_STANDINGS(

    driverStandingsId SERIAL PRIMARY KEY,

    raceId INT REFERENCES public.RACES(raceId) NOT NULL,

    driverId INT REFERENCES public.DRIVERS(driverId) NOT NULL,

    points FLOAT,

    "position" INT,

    wins INT

);

DROP TABLE IF EXISTS public.CONSTRUCTOR\_STANDINGS CASCADE;

CREATE TABLE public.CONSTRUCTOR\_STANDINGS(

    constructorStandingsId SERIAL PRIMARY KEY,

    raceId INT REFERENCES public.RACES(raceId) NOT NULL, -- FK

    constructorId INT REFERENCES public.CONSTRUCTORS(constructorId) NOT NULL, -- FK

    points FLOAT,

    "position" INT,

    wins INT

);

**DML Statements**

The following statements populate the above schema with data from the cleaned data

files:

-- Seeding data through csv file

-- Seeding data into public.drivers table

COPY public.drivers

FROM 'D:/Docs/Masters/CIS\_556/data/drivers.csv'

DELIMITER ','

CSV HEADER;

-- Seeding data into public.CONSTRUCTORS table

COPY public.CONSTRUCTORS

FROM 'D:/Docs/Masters/CIS\_556/data/constructors.csv'

DELIMITER ','

CSV HEADER;

-- Seeding data into public.CIRCUITS table

COPY public.CIRCUITS(circuitId, circuitRef, "name", "location", country, lat, lng, url)

FROM 'D:/Docs/Masters/CIS\_556/data/circuits.csv'

DELIMITER ','

CSV HEADER;

-- Seeding data into public.SEASONS table

COPY public.SEASONS

FROM 'D:/Docs/Masters/CIS\_556/data/seasons.csv'

DELIMITER ','

CSV HEADER;

-- Seeding data into public.STATUS table

COPY public.STATUS

FROM 'D:/Docs/Masters/CIS\_556/data/status.csv'

DELIMITER ','

CSV HEADER;

-- Seeding data into public.RACES table

COPY public.RACES(raceId, "year", round, circuitId, "name", "date", "time", url)

FROM 'D:/Docs/Masters/CIS\_556/data/races.csv'

DELIMITER ','

CSV HEADER;

-- Seeding data into public.PITSTOPS table

COPY public.PITSTOPS(raceid, driverid, stop, lap, "time", duration)

FROM 'D:/Docs/Masters/CIS\_556/data/pit\_stops.csv'

DELIMITER ','

CSV HEADER;

-- Seeding data into public.LAPTIMES table

COPY public.LAPTIMES

FROM 'D:/Docs/Masters/CIS\_556/data/lap\_times.csv'

DELIMITER ','

CSV HEADER;

-- Seeding data into public.RESULTS table

COPY public.RESULTS

FROM 'D:/Docs/Masters/CIS\_556/data/results.csv'

DELIMITER ','

CSV HEADER;

-- Seeding data into public.DRIVER\_STANDINGS table

COPY public.DRIVER\_STANDINGS

FROM 'D:/Docs/Masters/CIS\_556/data/driver\_standings.csv'

DELIMITER ','

CSV HEADER;

-- Seeding data into public.CONSTRUCTOR\_STANDINGS table

COPY public.CONSTRUCTOR\_STANDINGS

FROM 'D:/Docs/Masters/CIS\_556/data/constructor\_standings.csv'

DELIMITER ','

CSV HEADER;

**Data Analysis Queries**

The following section contains several queries that we have run on the data for gaining deep insights about the F1

--1. Analyzing Drivers' Failure to Complete Races

SELECT distinct d.driverRef, d.driverid, COUNT(\*) AS failed\_races

FROM results r

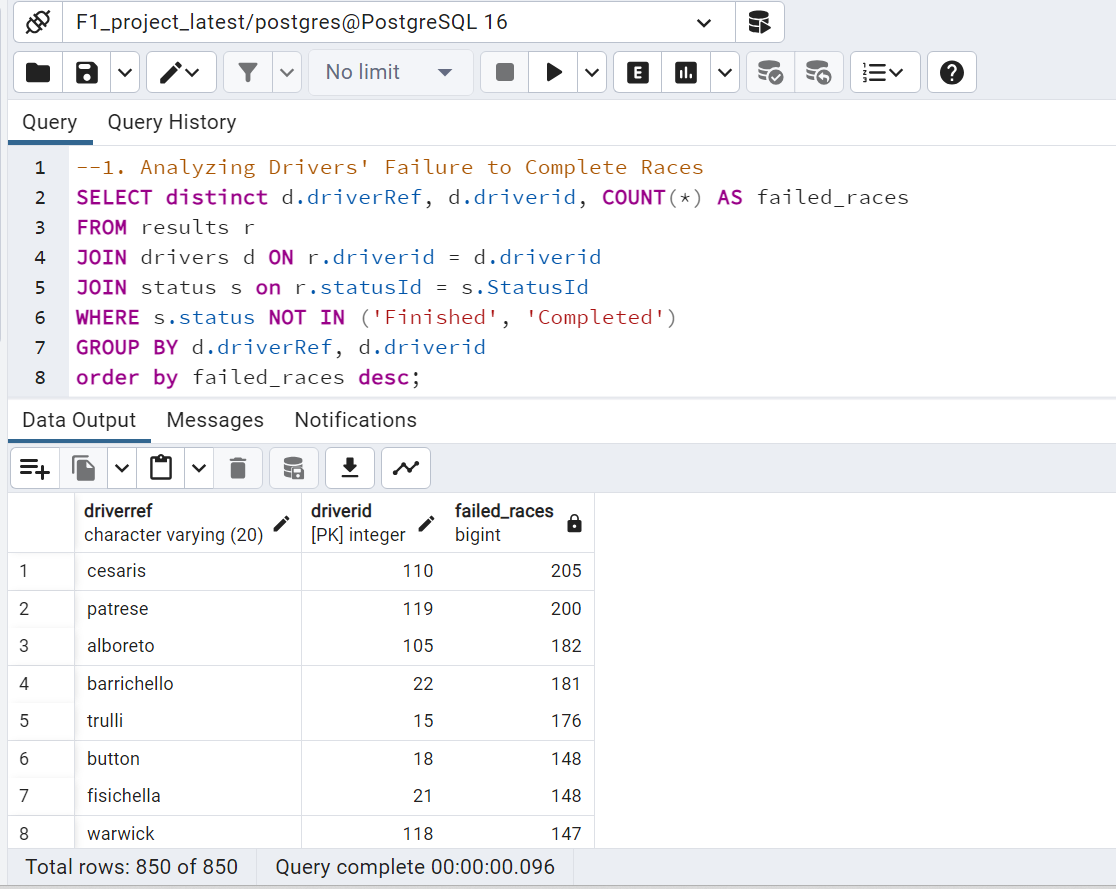
JOIN drivers d ON r.driverid = d.driverid

JOIN status s on r.statusId = s.StatusId

WHERE s.status NOT IN ('Finished', 'Completed')

GROUP BY d.driverRef, d.driverid

order by failed\_races desc;



--2. Evaluating the Impact of Circuit Characteristics on Race Outcomes:

SELECT c.name AS circuit\_name, d.driverRef, AVG(r.position) AS average\_position

FROM results r

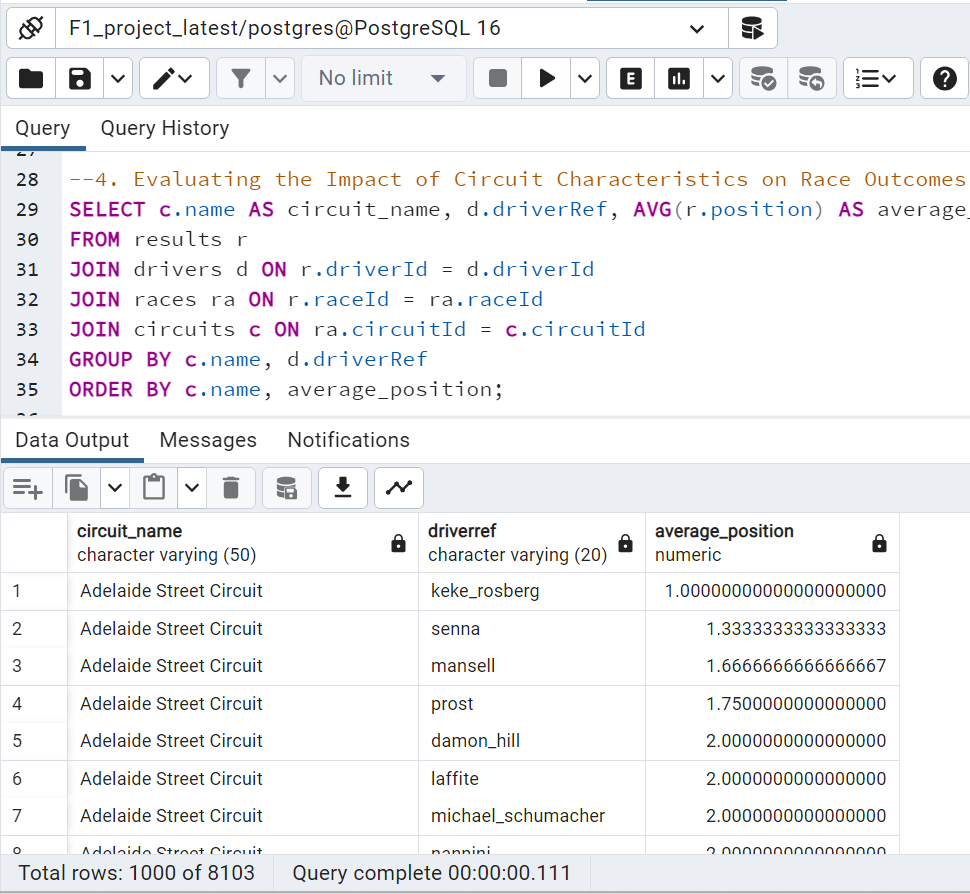
JOIN drivers d ON r.driverId = d.driverId

JOIN races ra ON r.raceId = ra.raceId

JOIN circuits c ON ra.circuitId = c.circuitId

GROUP BY c.name, d.driverRef

ORDER BY c.name, average\_position;



--3. Studying the Effects of Pit Stop Strategies on Race Results:

SELECT d.driverRef, ra.year, AVG(p.duration) AS average\_pitstop\_duration, AVG(r.position) AS

average\_race\_position

FROM pitstops p

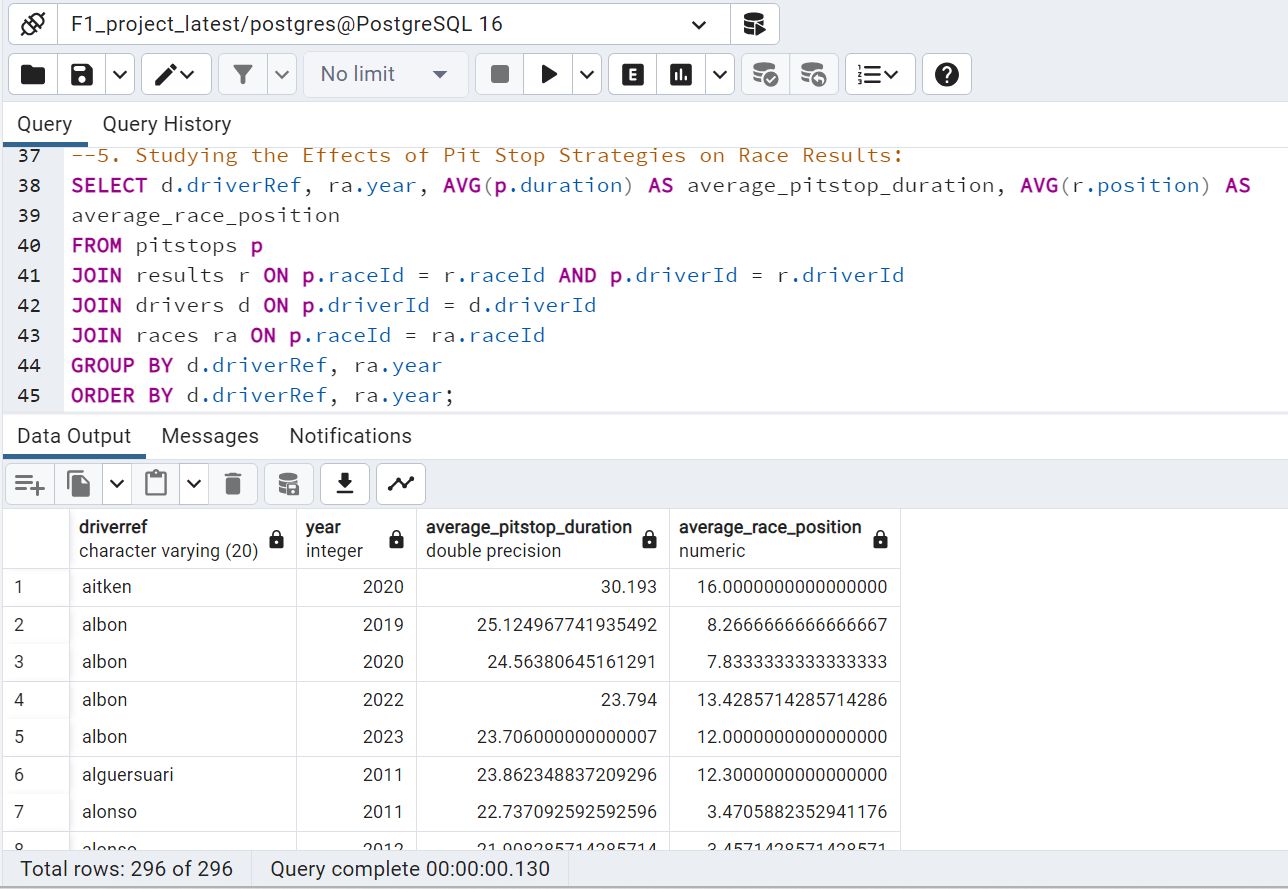
JOIN results r ON p.raceId = r.raceId AND p.driverId = r.driverId

JOIN drivers d ON p.driverId = d.driverId

JOIN races ra ON p.raceId = ra.raceId

GROUP BY d.driverRef, ra.year

ORDER BY d.driverRef, ra.year;



--4. Uncovering Trends and Patterns Behind Drivers' Failure to Complete Races:

SELECT dr.driverRef, COUNT(\*) AS failure\_count

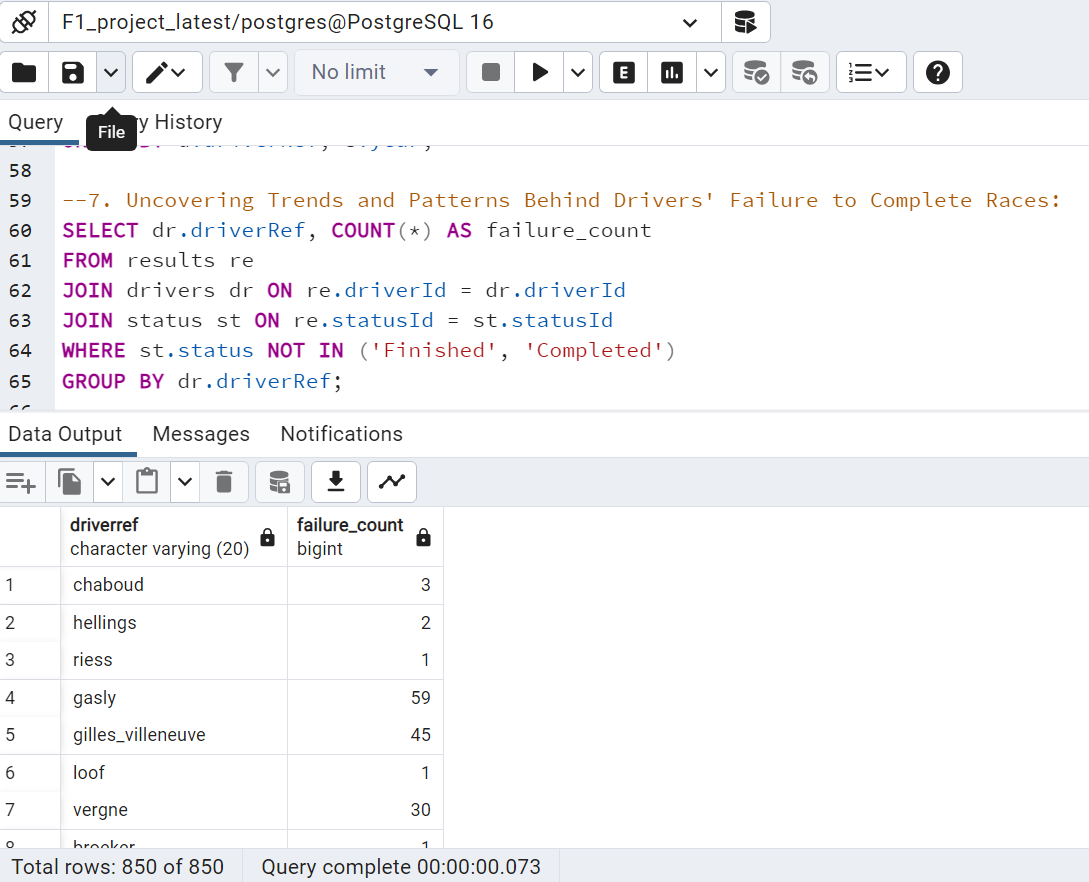
FROM results re

JOIN drivers dr ON re.driverId = dr.driverId

JOIN status st ON re.statusId = st.statusId

WHERE st.status NOT IN ('Finished', 'Completed')

GROUP BY dr.driverRef;



-- 5.Driver Performance Analysis Across Different Circuits: This analysis could reveal how

-- drivers perform on different types of circuits (e.g., street vs. race tracks)

SELECT dr.driverRef, ci.name AS circuit\_name, Round(AVG(re.position)) AS average\_position

FROM results re

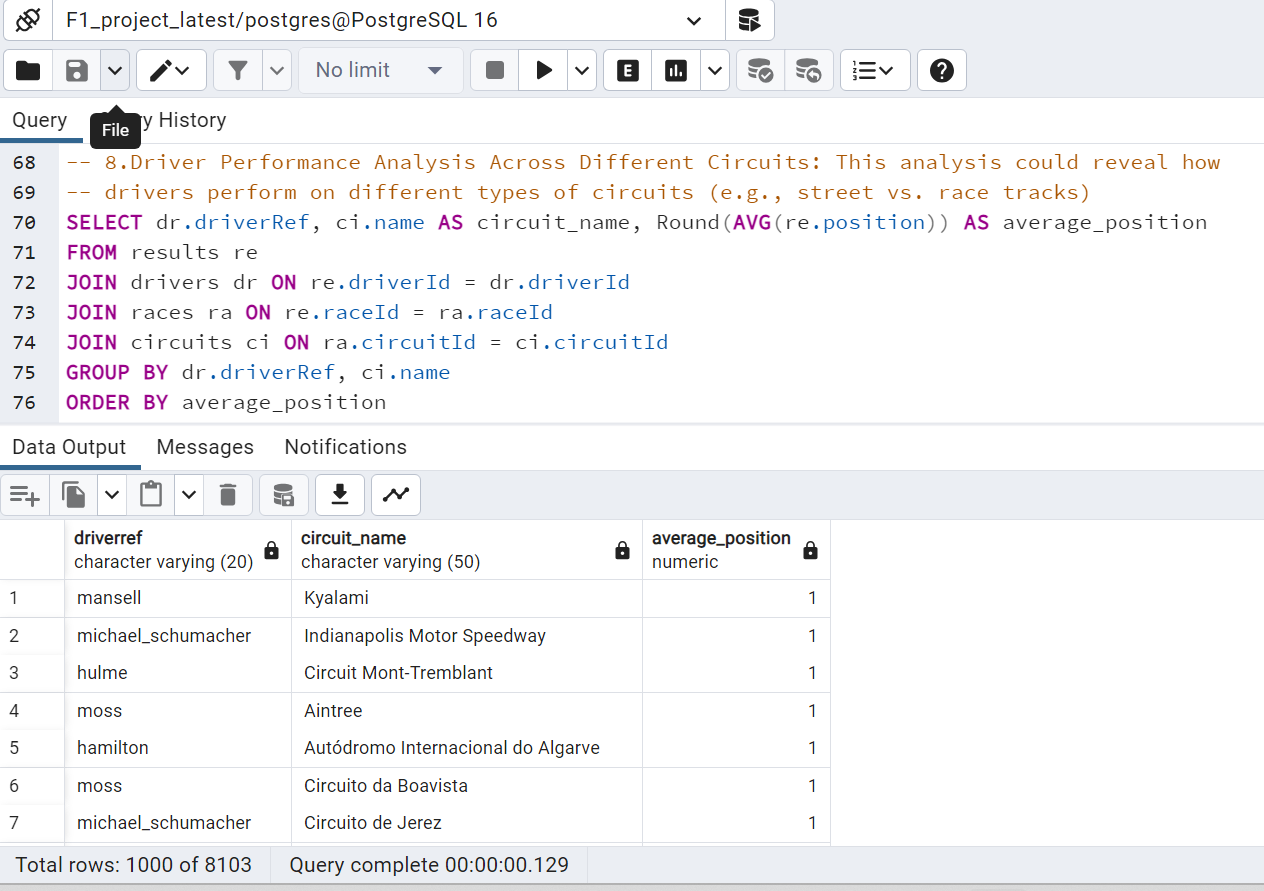
JOIN drivers dr ON re.driverId = dr.driverId

JOIN races ra ON re.raceId = ra.raceId

JOIN circuits ci ON ra.circuitId = ci.circuitId

GROUP BY dr.driverRef, ci.name

ORDER BY average\_position



-6. Impact of Pit Stops on Race Outcomes: Investigate how the number and duration of pit

--  stops affect a driver's final position in a race.

SELECT re.raceId, dr.driverRef, COUNT(ps.stop) AS pit\_stop\_count, AVG(ps.duration) AS

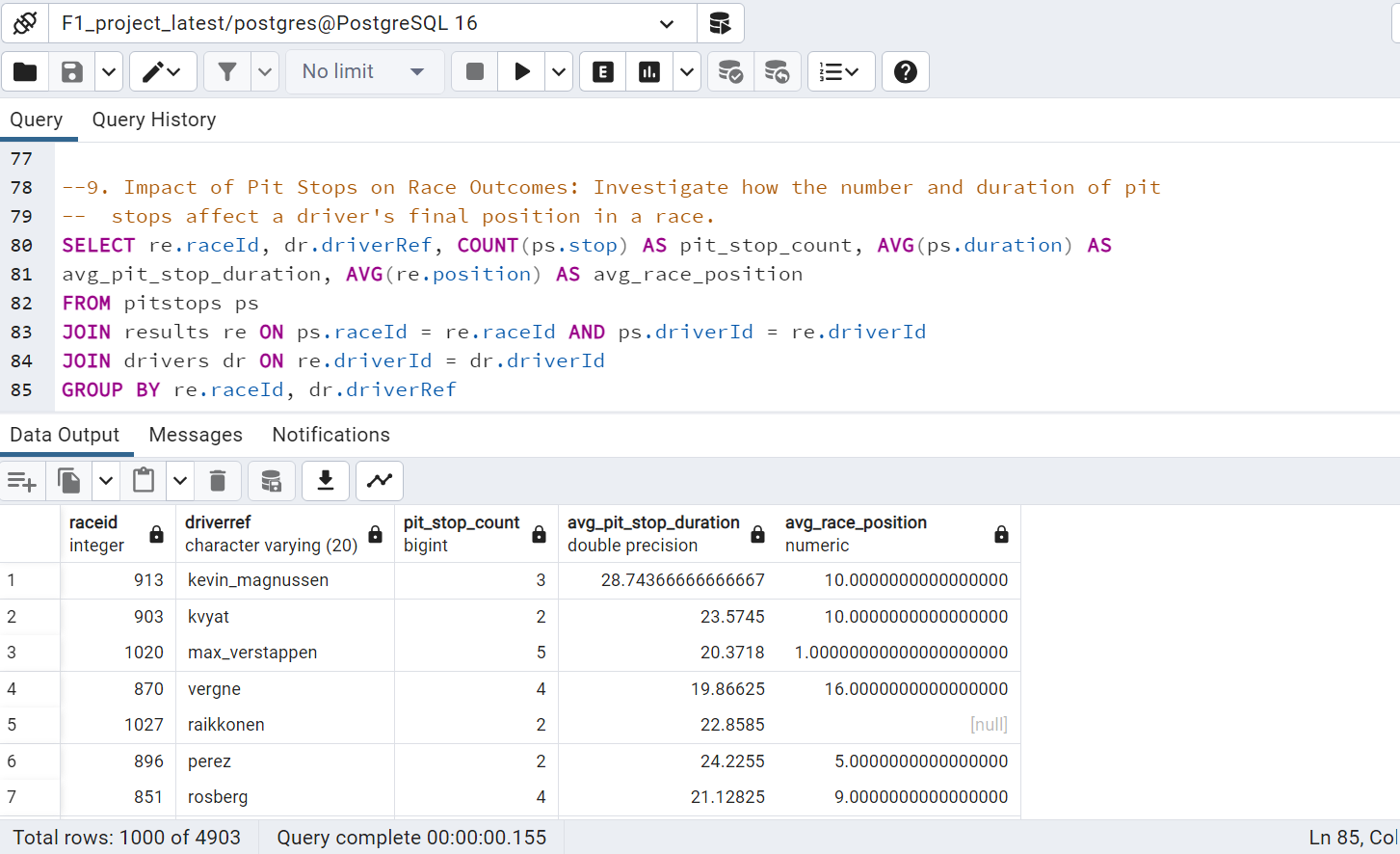
avg\_pit\_stop\_duration, AVG(re.position) AS avg\_race\_position

FROM pitstops ps

JOIN results re ON ps.raceId = re.raceId AND ps.driverId = re.driverId

JOIN drivers dr ON re.driverId = dr.driverId

GROUP BY re.raceId, dr.driverRef



--7. Seasonal Performance Trends of Drivers: Analyze how drivers' performances vary across

-- different seasons.

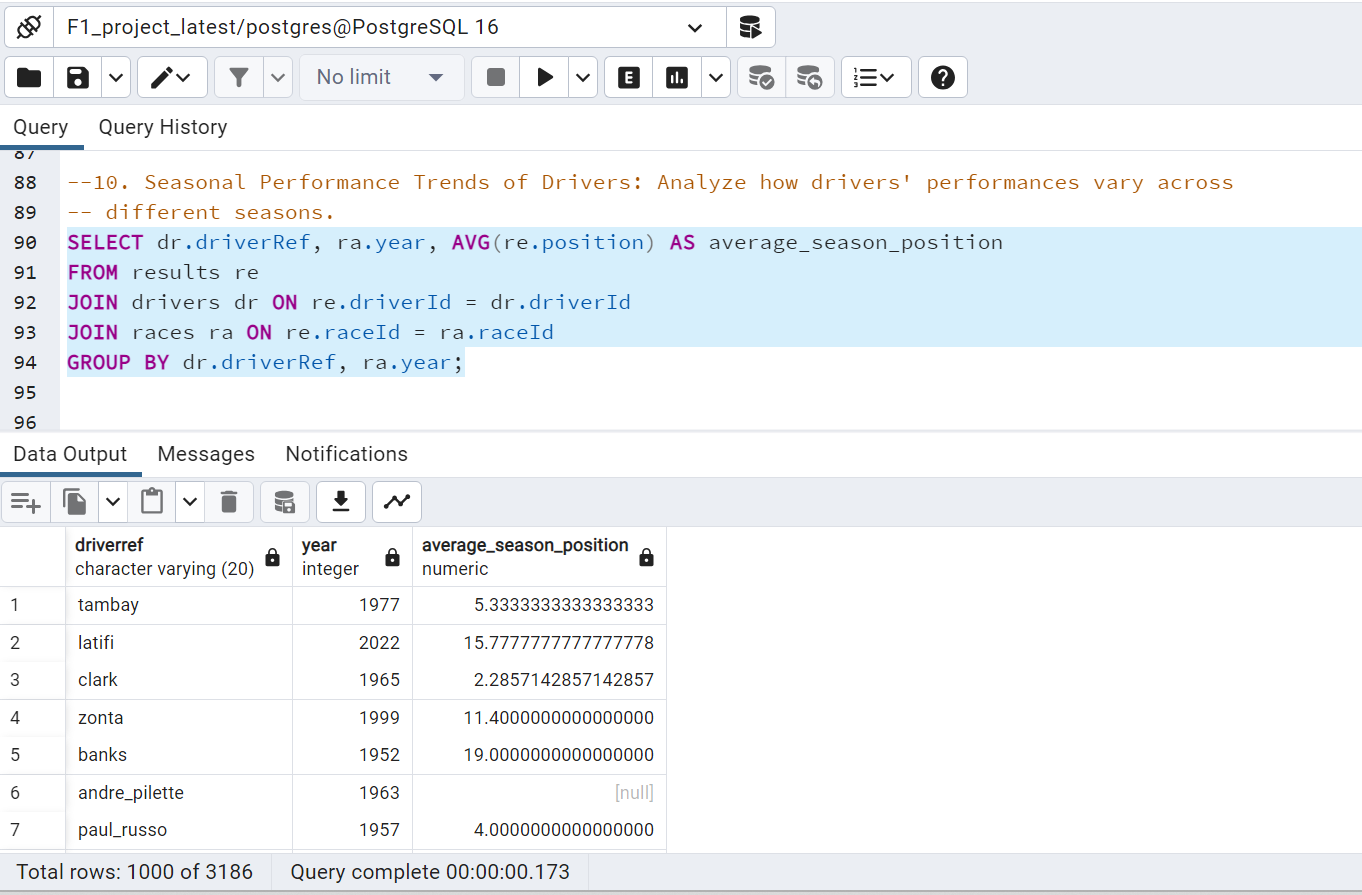
SELECT dr.driverRef, ra.year, AVG(re.position) AS average\_season\_position

FROM results re

JOIN drivers dr ON re.driverId = dr.driverId

JOIN races ra ON re.raceId = ra.raceId

GROUP BY dr.driverRef, ra.year;



--8.Constructor Team Performance Analysis: Assess the overall performance of constructor

-- teams over various seasons.

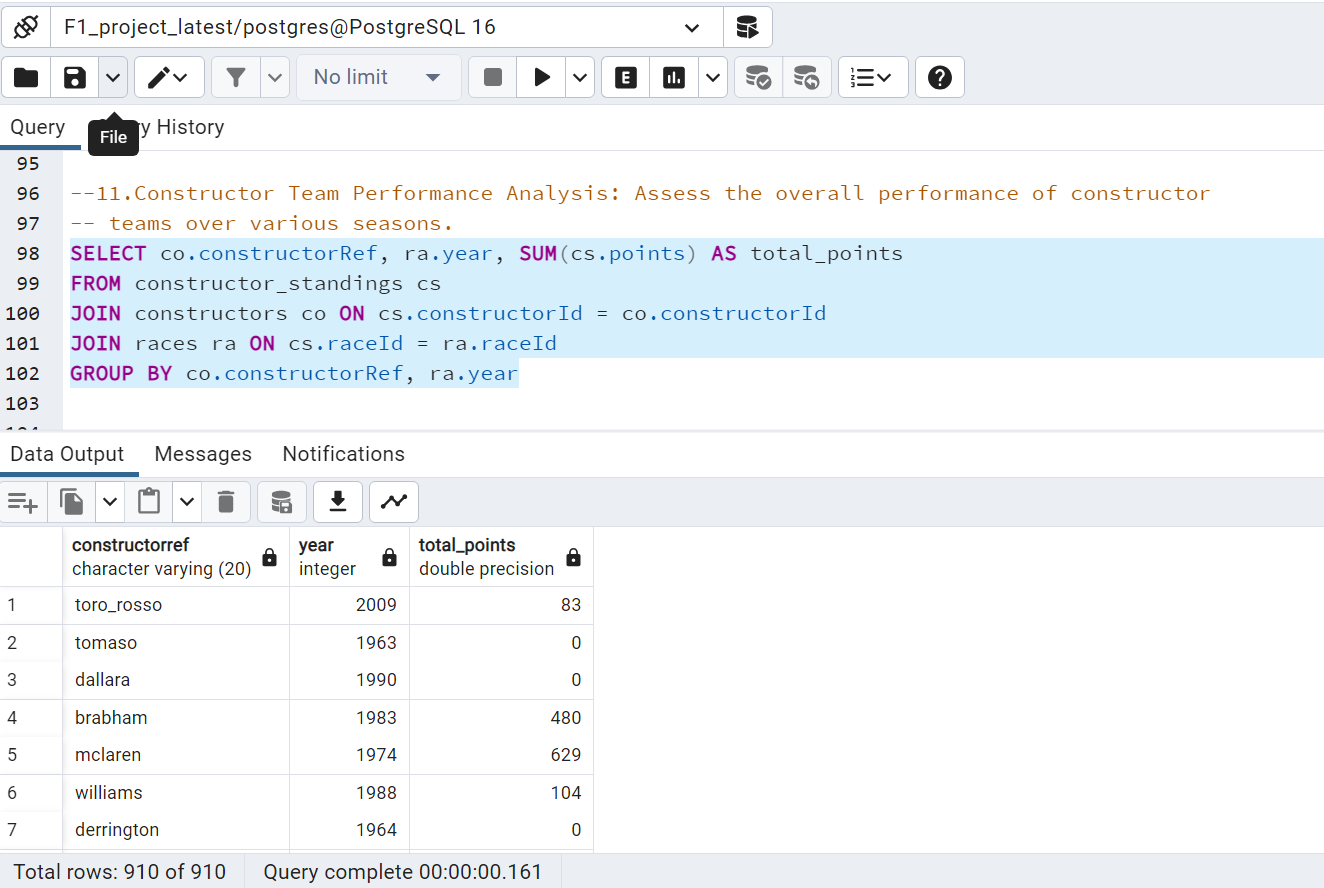
SELECT co.constructorRef, ra.year, SUM(cs.points) AS total\_points

FROM constructor\_standings cs

JOIN constructors co ON cs.constructorId = co.constructorId

JOIN races ra ON cs.raceId = ra.raceId

GROUP BY co.constructorRef, ra.year



--9.Analysis of Lap Times Across Different Races: Examine trends in lap times for drivers across

-- different races to gauge consistency and improvement.

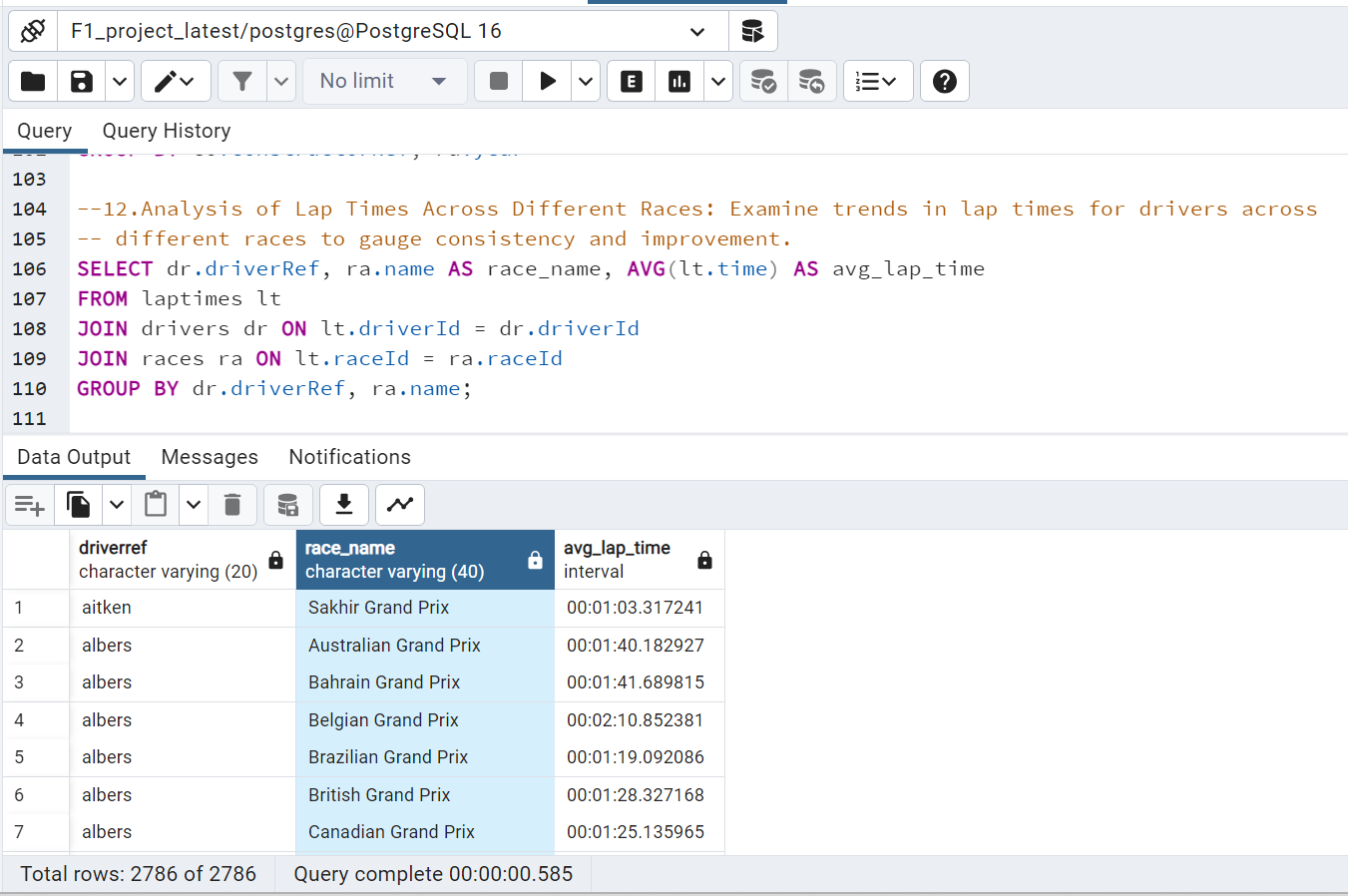
SELECT dr.driverRef, ra.name AS race\_name, AVG(lt.time) AS avg\_lap\_time

FROM laptimes lt

JOIN drivers dr ON lt.driverId = dr.driverId

JOIN races ra ON lt.raceId = ra.raceId

GROUP BY dr.driverRef, ra.name;



--10. Driver Standings and Win Analysis Over Years: Look into how drivers' standings and win

-- rates have evolved over the years.

SELECT dr.driverRef, r.year, AVG(ds.position) AS average\_position, SUM(ds.wins) AS

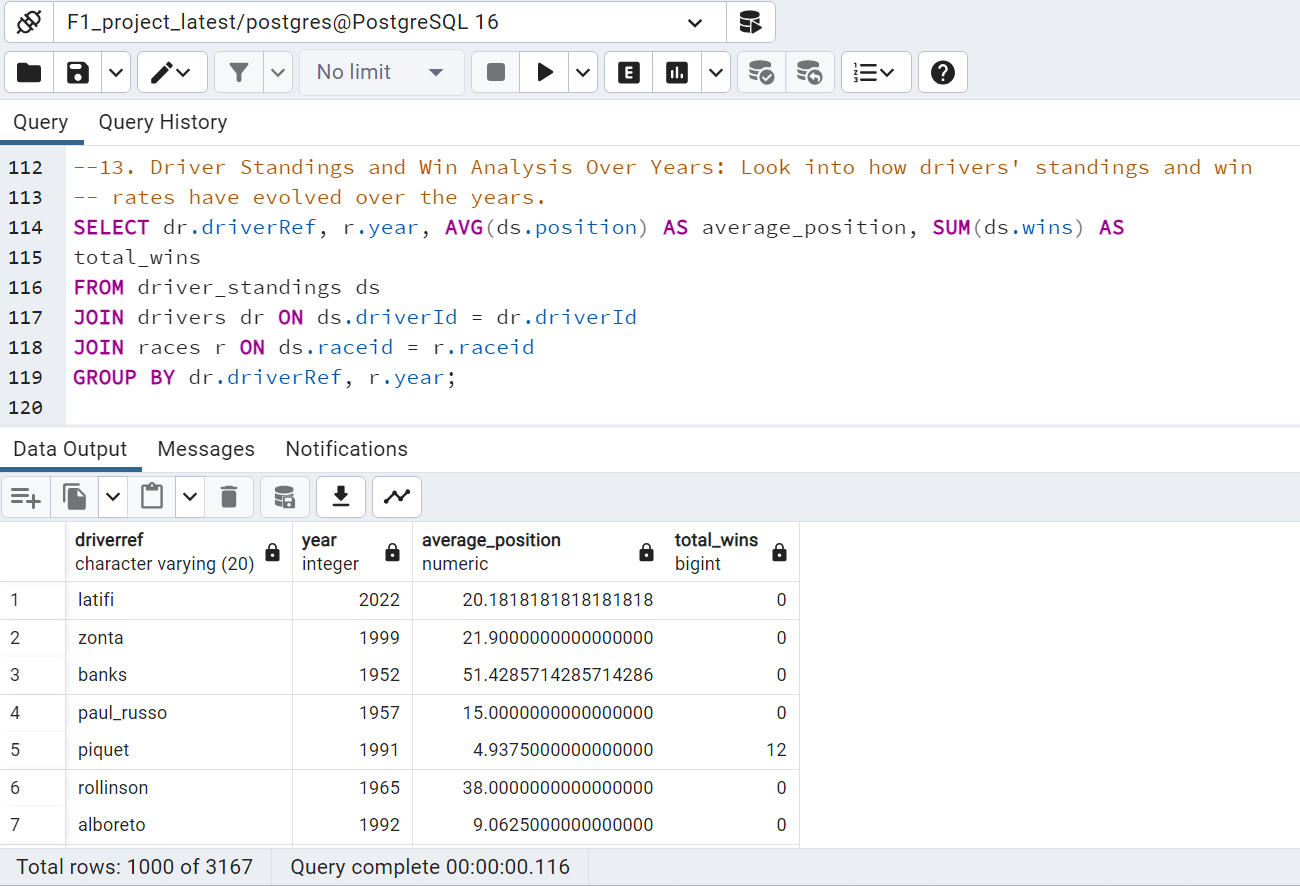
total\_wins

FROM driver\_standings ds

JOIN drivers dr ON ds.driverId = dr.driverId

JOIN races r ON ds.raceid = r.raceid

GROUP BY dr.driverRef, r.year;



--11. Total Wins of the drivers

SELECT drivers.forename AS first\_name, drivers.surname AS last\_name,

drivers.dob AS birth\_date, COUNT(\*) AS Total\_Wins

FROM results

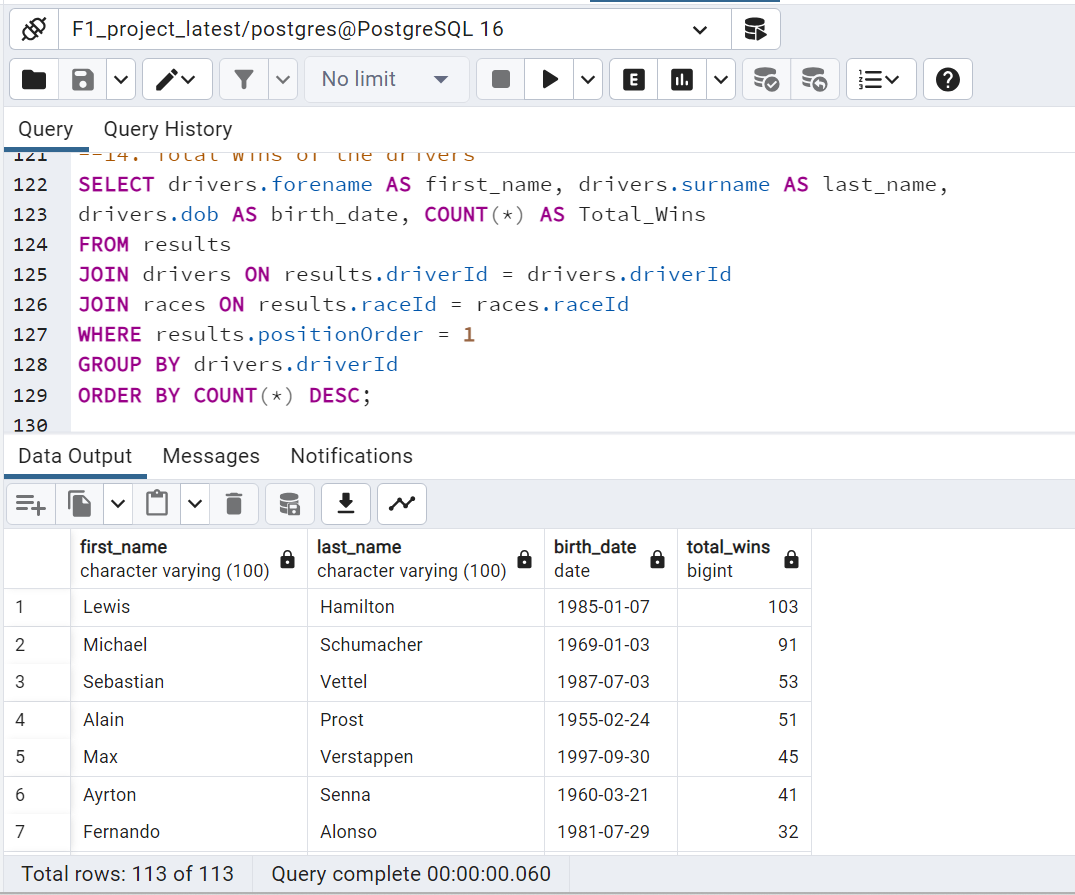
JOIN drivers ON results.driverId = drivers.driverId

JOIN races ON results.raceId = races.raceId

WHERE results.positionOrder = 1

GROUP BY drivers.driverId

ORDER BY COUNT(\*) DESC;



--12. Comparing Constructor Performances Over Seasons: query improvement

;with sns as (

    Select Distinct "year"

    from races

    Order by "year"

)

SELECT c.constructorRef, s.year, SUM(cs.points) AS total\_points, c.name

FROM constructor\_standings cs

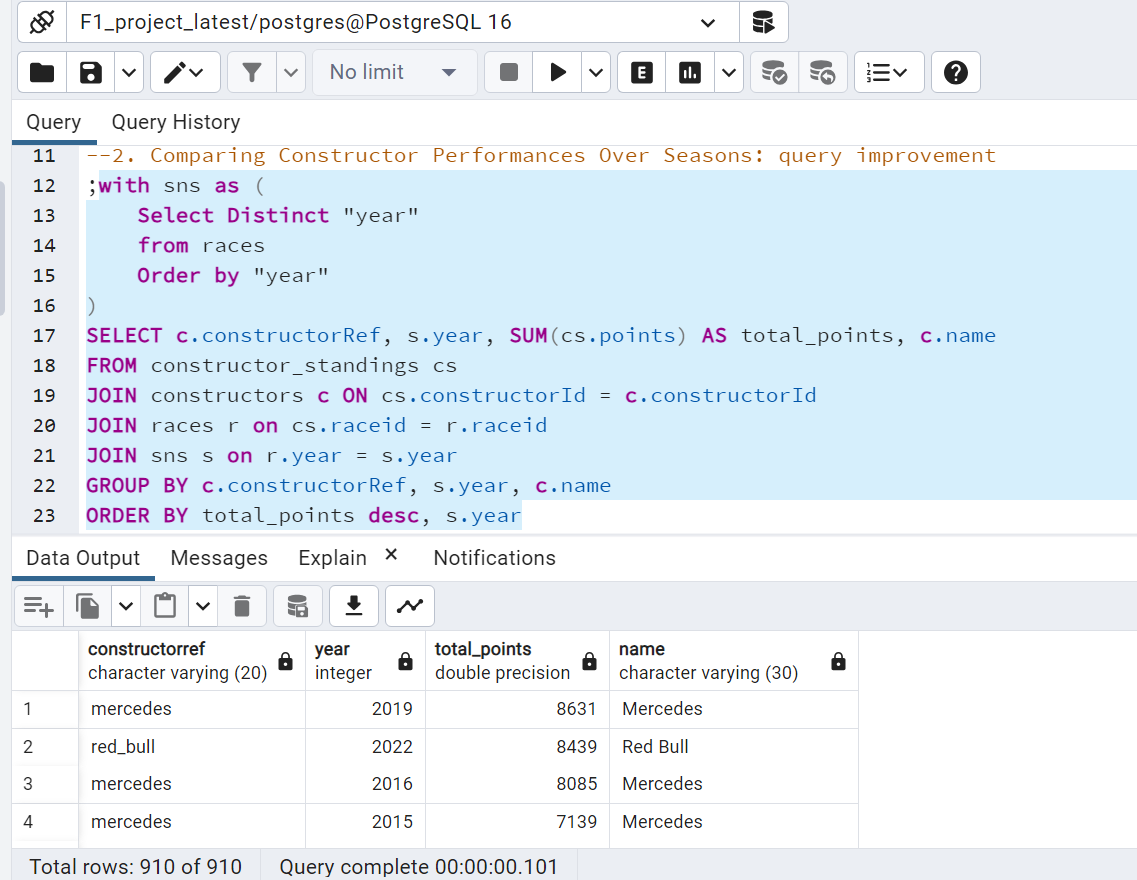
JOIN constructors c ON cs.constructorId = c.constructorId

JOIN races r on cs.raceid = r.raceid

JOIN sns s on r.year = s.year

GROUP BY c.constructorRef, s.year, c.name

ORDER BY total\_points desc, s.year



**Index Implementation**

A set of indexes was created to optimize queries on the dataset based on standard fields used more often for sorting and grouping. These fields were identified by analyzing sample queries and anticipating the most common types of queries that will be run on the dataset. This approach is expected to improve query performance and make data retrieval more efficient:

CREATE INDEX idx\_constructorRef ON constructors (constructorRef);

CREATE INDEX idx\_name ON constructors("name");

CREATE INDEX idx\_driverref ON drivers(driverRef);

CREATE INDEX idx\_forename\_surname ON drivers(forename, surname);

CREATE INDEX idx\_dob ON drivers(dob);

CREATE INDEX idx\_time ON laptimes("time");

CREATE INDEX idx\_time\_pitstop ON pitstops("time");

**Performance Evaluation:**

Creating indexes has proven effective in enhancing buckets' performance and memory allocation, specifically in the recursive query (last query). However, while the benefits of indexes may take time to become apparent in simpler setups, they can be beneficial when dealing with more complex queries. In such scenarios, indexes can significantly expedite data retrieval, making them an invaluable tool for optimized performance. Below is the query plan executed by enabling EXPLAIN ANALYZE [2] metrics in PostgreSQL

**Before Index:**



**After Index:**



**Final Conclusions**

As part of our project, we focused on organizing the data into a schema by implementing it in PostgreSQL. During this process, we executed sample queries against the database and observed various factors related to the Formula 1 World Championship. Specifically, we looked at circuit conditions, driver performances at particular circuits, and improvements made over the years. As we continued our work, we identified a few unnecessary items that normalisation could remove. Normalization is organizing data in a database to be consistent and easy to manage. By performing normalization, we reduced the size of our database and streamlined the data management process. Furthermore, we demonstrated the importance of adding indexes to the database. Indexes help to improve memory allocation and execution time, making the database more efficient and effective. Our team successfully integrated indexes into the database, significantly improving memory allocation and execution time.

**Reference**

[1] <https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020>

[2] <https://www.postgresql.org/docs/current/using-explain.html#USING-EXPLAIN-ANALYZE>