****

**COPMXP323(21A) - Project**

**Group members:**  Al Zhang, Joshua(Taohua /wu) and Connor Welham

**DEMONSTRATION DATE : Thursday 3rd June, 11.20am**

**Milestone I**……………………………………...4-7

Entities………………………………………………….4

Relationship…………………………………………….5

ER Diagram…………………………………………….6

Assumptions…………………………………………….6

Oracle Test……………………………………………...7

**Milestone II** …………………………………… 8-24

Improved ER Diagram And Relation Scheme……....8

Relations in Oracle …………………………………….8

Small Dataset…………………………………………..9-10

Large Dataset…………………………………………..11-12

SQL Tables in Oracle………………………………….13-16

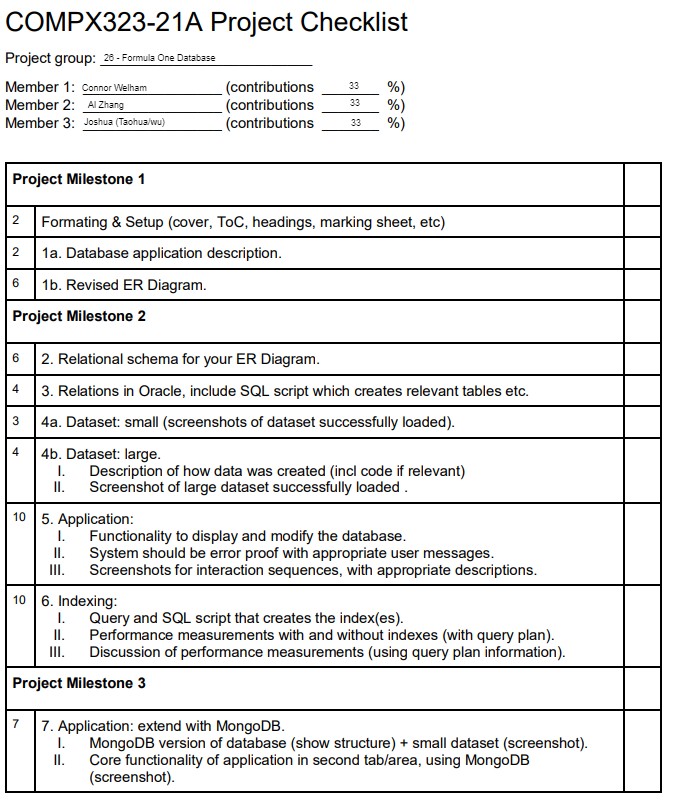
Basic Application……………………………………...17-20

Indexes………………………………………….……...21-24

**Milestone III** …………………………………..25-28

MongoDB Collections…………………..…………..25-26

MongoDB C# program……………………………..27-28

****

**Introduction**

The Fédération Internationale de l'Automobile (FIA) have asked us to create a database for the 2020 Formula One Season.

**Entities**

Team

The Formula One Championship is made up of many teams. Every team has a unique name due to copyright. The FIA also wants us to track where the teams are based, who is the founder and how many championships they have won.

Person

A driver is a person and a founder is a person. A person has a unique id number. We also want to store their name and their date of birth.

Driver

The drivers are the people who race. Every driver has a unique number which is displayed on their car. We have been asked to track their 2020 final position, name, date of birth and number of championships won. They also take all attributes from a person as they are a person.

Founder

The people who have founded teams. They take all attributes from a person as they are a person. Founder is a weak entity as we do not want to hold their information if the team they created is removed from the database.

Car

Cars play a huge factor in success in Formula One. Each car has a unique name. We want to keep track of the power unit, tyres, date it was created and who designed the car. Car is a weak entity because we do not want to hold their information if the team who manufactured the car is removed from the database.

Grand Prix

Grand Prix’s are the foundations of Formula One. Each Grand Prix marks a stage in the championship. Each Grand Prix is held in a unique city in a unique country. The FIA wanted us to track when it was first held, the circuit length and the number of laps.

Country

As each Grand Prix is held in a different country, FIA wants us to hold some data about each country so that the commentators can speak about them. Every country has a unique name and we want to store the population and the area.

**Relationships**

Sign (1-Many)

Each driver is signed by a team. A driver can only be signed to one team, but may be unsigned. A team can have several members, but must have at least one driver in their team to be recorded in our database. When a driver signs for a team, they receive a salary and are on contract for a certain number of years.

Manufacturers (1-Many)

Each car is manufactured by the team. They are specially built for their drivers and to maximise performance. Each team may have several cars. Teams do not collaborate on car manufacturing. A car is a weak entity because we will not record information of a car if it is not manufactured by a team. If a team disbands, we will stop recording their cars as it will not be driven.

Participates\_In (Many-Many)

Each driver participates in Grand Prixs. Drivers do not have to participate in every Grand Prix, but usually aim to participate in all of them if they want a high placing. Grand Prix’s are made up of multiple drivers. We also want to store the position the drivers come in each Grand Prix so we can determine who is the winner. In addition, we want to store the date the Grand Prix started and ended.

Held\_At (1-1)

Each Grand Prix is held in a unique country and only one country. A country can not hold more than one Grand Prix.

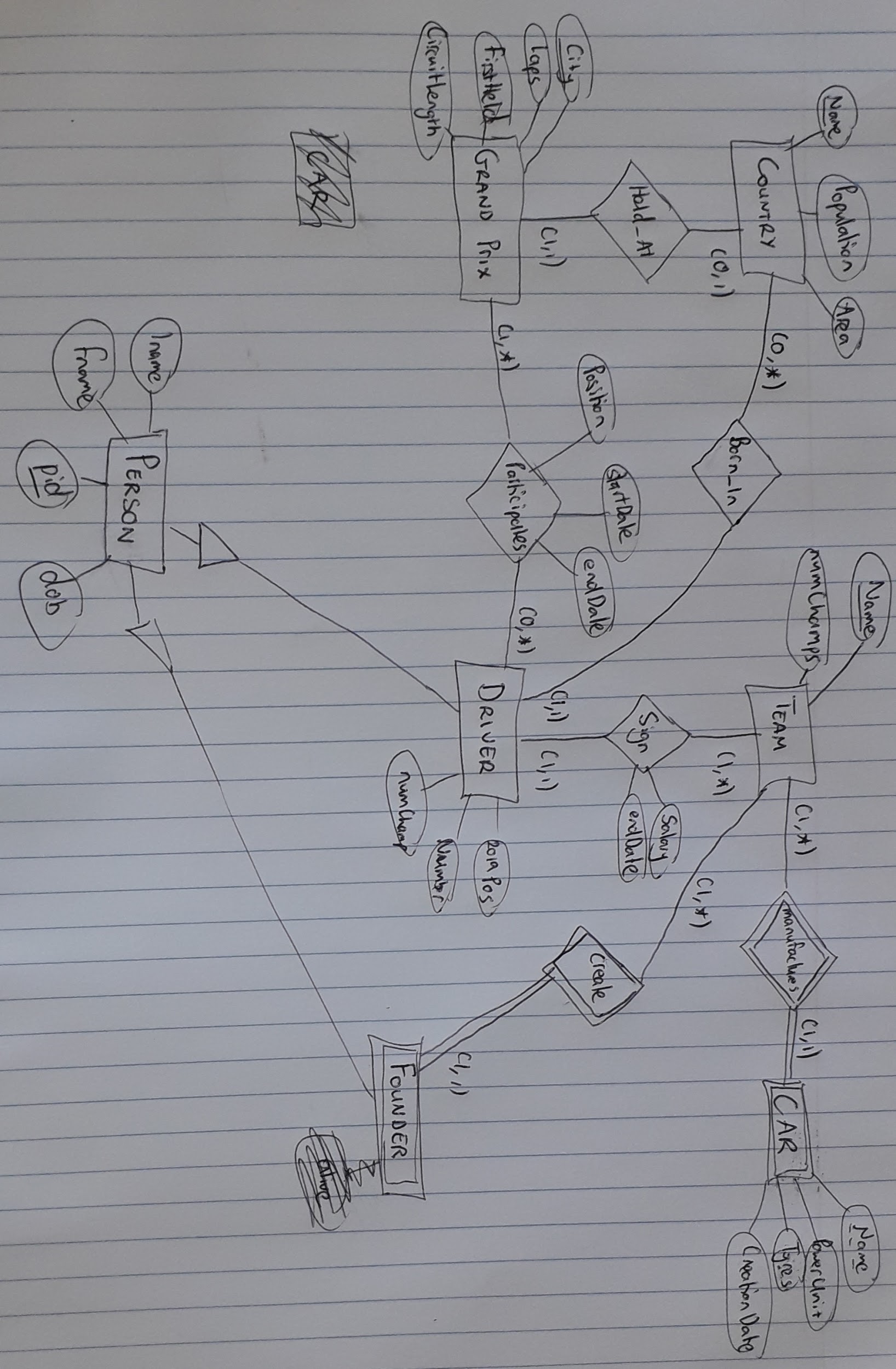
Born\_In (1-Many)

Each driver is born in one country. Countries produce multiple drivers. In many cases, multiple drivers all come from the same country due to the culture of the country.

Creates(1-Many)

A founder creates a team. A founder will only create one team but a team can be created by many founders.

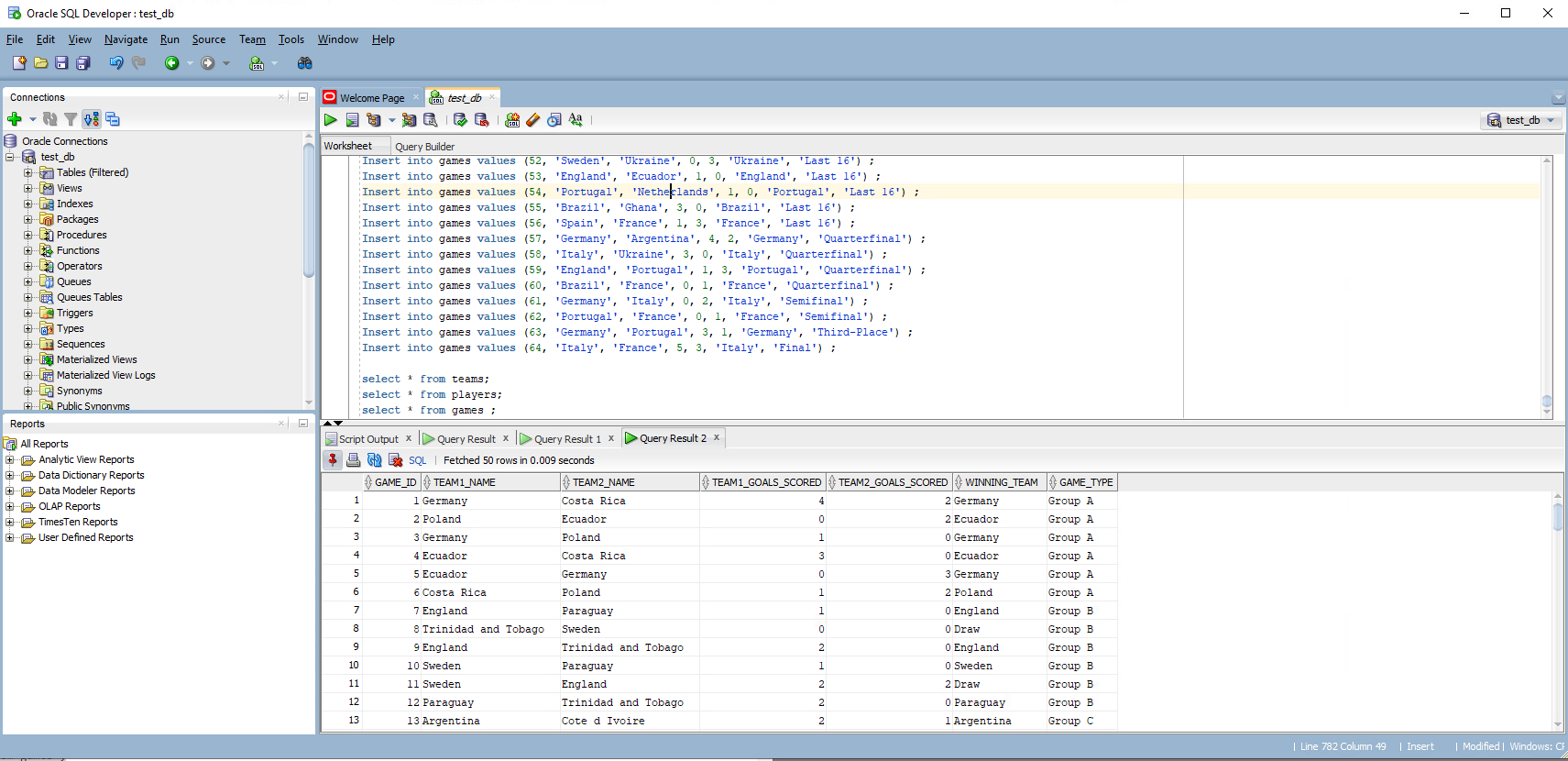
**ER Diagram**

****

Assumptions

* Each driver drives a car from the team that manufactured the car
* Data is not stored about which driver’s drive which car, rather the database is focused on storing personal information about the driver/team and how they have placed in the Formula One season
* Storing the start date of the driver’s contract is unnecessary and we only want to show when their contract expires
* The database is made for the 2020 season and therefore stores the driver’s position in 2019, but can be used for any season and this attribute can be left null
* A Grand Prix is identified by which city they are in
* Salary is recorded in millions of USD

**Oracle Test**



Connection Name: COMPX323 F1 Test1

Group Username: COMPX323\_26

Password: DasqWQBhbR

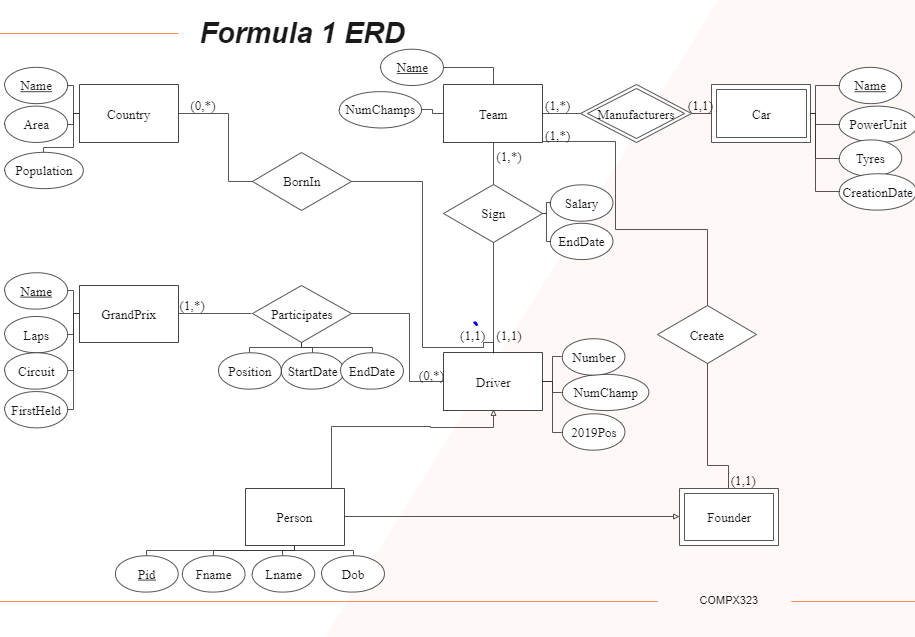
Hostname: oracle.cms.waikato.ac.nz

Port: 1521

Service Name: teaching

**Introduction**

**ER Diagram - Final**

****

Changes:

* GrandPrix has a key of name rather than city as grand prixs can be held in the same city more than once.
* Used edrawmax.com as the paper copy had some attributes that were hard to see
* Also note that name under the entity car should be underlined with a dotted line as it is a weak entity

**Relational Schema**

Country(countryName, population, area)

Driver(pid, fname, name, dob, numChamp, number, pastPos, countryName, teamName, salary, endDate)

Team(teamName, numChamp)

GrandPrix(prixName, laps, circuitLength, firstHeld, countryName)

Founder(pid, teamName, fname, lname, dob)

Car(carName, teamName, powerUnit, tyres, creationDate)

Participates(prixName, pid, position, pdate)

**First Relations**

To start creating our database system, we had decided to first start by collecting and entering data for the relationship ‘driver participates in Grand Prix(s)’. In order for us to create this relationship, we need to collect and enter data for the following tables:

* Driver
* Participates
* Grand Prix
* Team
* Country

Note that we have to create tables for team and country so that we can create the tables of driver and Grand Prix. However, we are not focusing on these tables so we will only create minimal data so that we can model the relationship.

**Test Dataset**

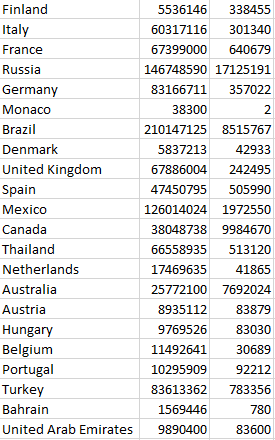
The test datasets were created so that they could be easily imported into Oracle. They are in excel/csv format and separated by commas.

**Small Dataset**

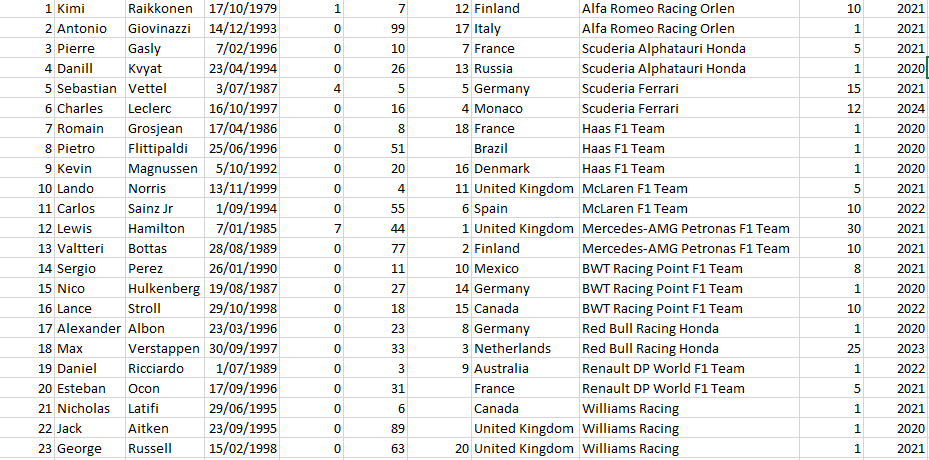
The small test datasets were collected from the Formula One 2020 season. The majority of this data was sourced from the following websites:

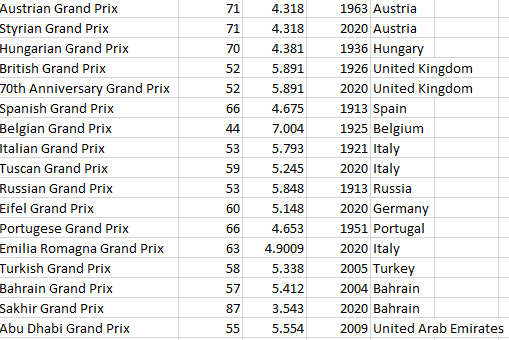
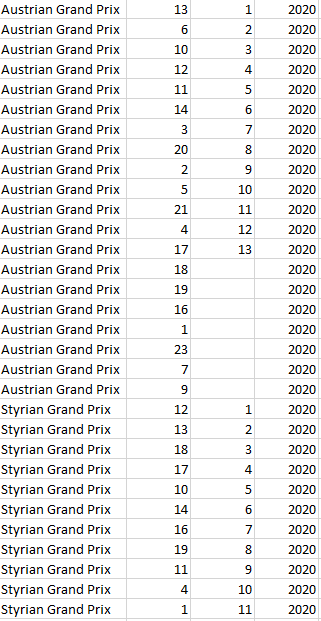
<https://en.wikipedia.org/wiki/2020_Formula_One_World_Championship>

<https://www.formula1.com/en/latest/article.2020-f1-grid-all-the-drivers-and-teams-racing.3UN6F7Q0mf7lVGzjv8l7ze.html>







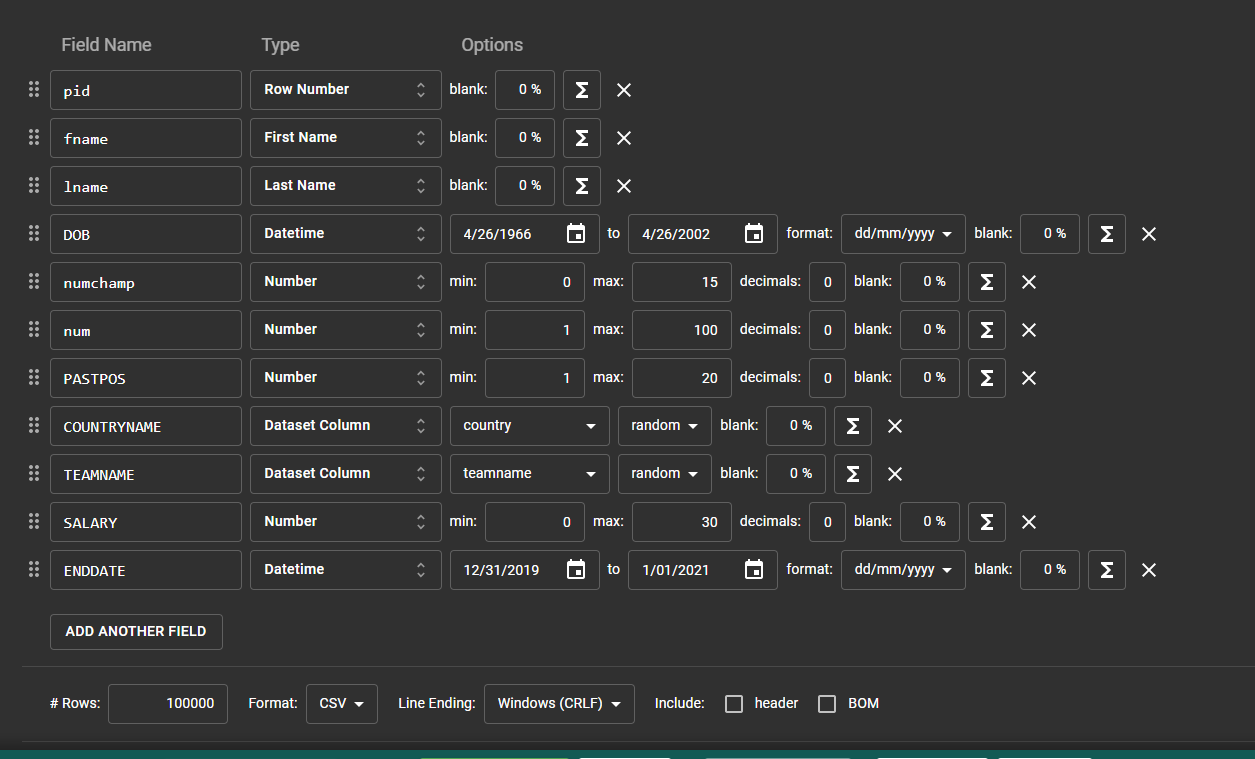


**Large Dataset**

The large datasets were generated by “Mockaroo”, which is a random data data generator tool.

The types and field names entered match the sql table. When you download a CSV file, every single attribute is random but meaningful. One CSV dataset has 300,000 driver data. We then could insert our data through Import CSV. To ensure each data can be inserted successfully, delimister, format and especially date, should be correct. Once the large dataset CSV was inserted, the table will be filled with a large amount of data. We can use the following query to check if the number is what we would expect: *select count(pid) from participates*

We encountered a few errors using Mockaroo. The first is that when importing our data, we would encounter a lot of errors. This was firstly due to a spelling mistake, but then we generated an error for violating constraints. What we found was that when we generated data randomly, it would duplicate some keys. To get round this issue, we ended up generating 9 csvs and then combining them into one later on. Each csv would focus on one Grand Prix and would have a data entry for each driver. This means that when we combined the csvs, we have one csv with 9 different entries for each driver.



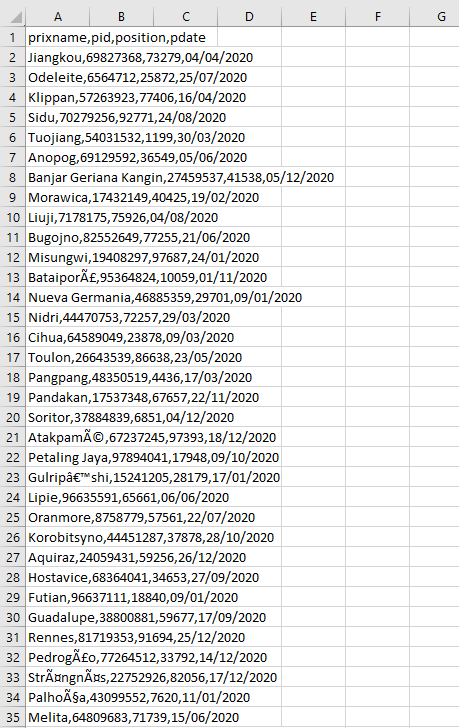
Number of Drivers in Database



Number of Participates in Database



CSV File contents



And continues on for several hundreds of thousand of data.

**SQL Tables**

*create table country*

*(*

*countryName varchar(20) not null,*

*population int not null,,*

*area int not null,,*

*primary key(countryName)*

*)*

*create table driver*

*(*

*pid smallint not null,*

*fname varchar(20) not null,,*

*lname varchar(20) not null,,*

*dob date not null,,*

*numChamp smallint not null,,*

*num smallint not null,,*

*pastPos smallint,*

*countryName varchar(30) not null,,*

*teamName varchar(30) not null,,*

*salary varchar(5) not null,,*

*endDate date not null,,*

*primary key(pid),*

*foreign key(countryName) references country,*

*foreign key(teamName) references team*

*)*

*create table team*

*(*

*teamName varchar(30) not null,*

*numChamp smallint not null,,*

*primary key(teamName)*

*)*

*--Changed primary key to name and removed city*

*create table grandPrix*

*(*

*name varchar(30) not null,*

*laps smallint not null,,*

*circuitLength varchar(8) not null,,*

*firstHeld date not null,,*

*countryName varchar(20) not null,,*

*primary key(name),*

*foreign key(countryName) references country*

*)*

*create table founder*

*(*

*pid smallint not null,*

*teamName varchar(30) not null,*

*fname varchar(20) not null,,*

*lname varchar(20) not null,,*

*dob date not null,,*

*primary key(pid, teamName),*

*foreign key(teamName) references team*

*)*

*create table car*

*(*

*carName varchar(20) not null,*

*teamName varchar(30) not null,*

*powerUnit varchar(20) not null,,*

*tyres varchar(20 not null,),*

*creationDate date not null,,*

*primary key(carName, teamName),*

*foreign key(teamName) references team*

*)*

*create table participates*

*(*

*prixName varchar(30) not null,*

*pid smallint not null,*

*position smallint,*

*pdate date not null,,*

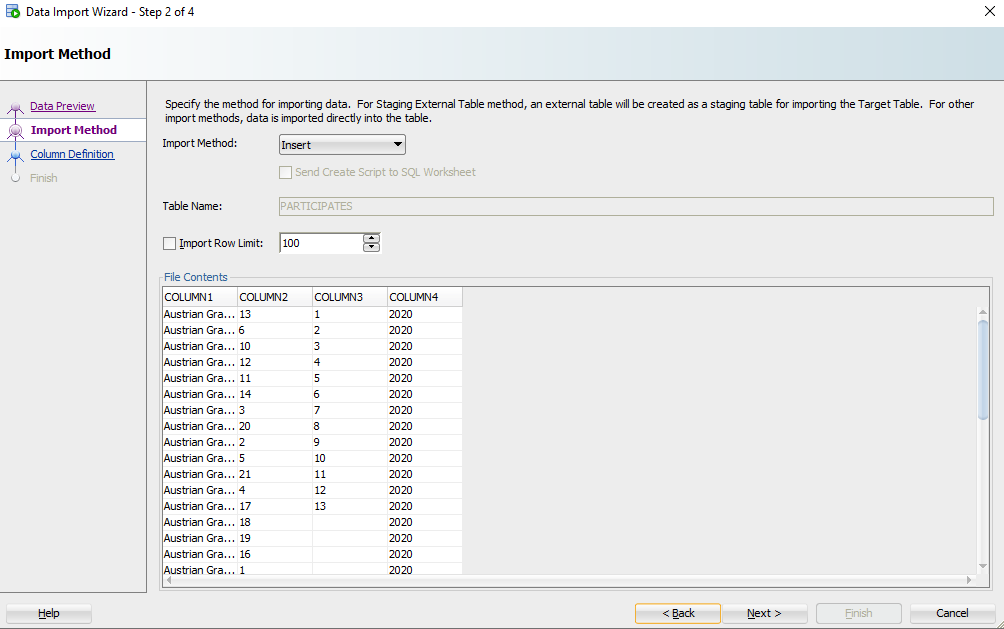
*primary key(prixName, pid),*

*foreign key(prixName) references grandPrix,*

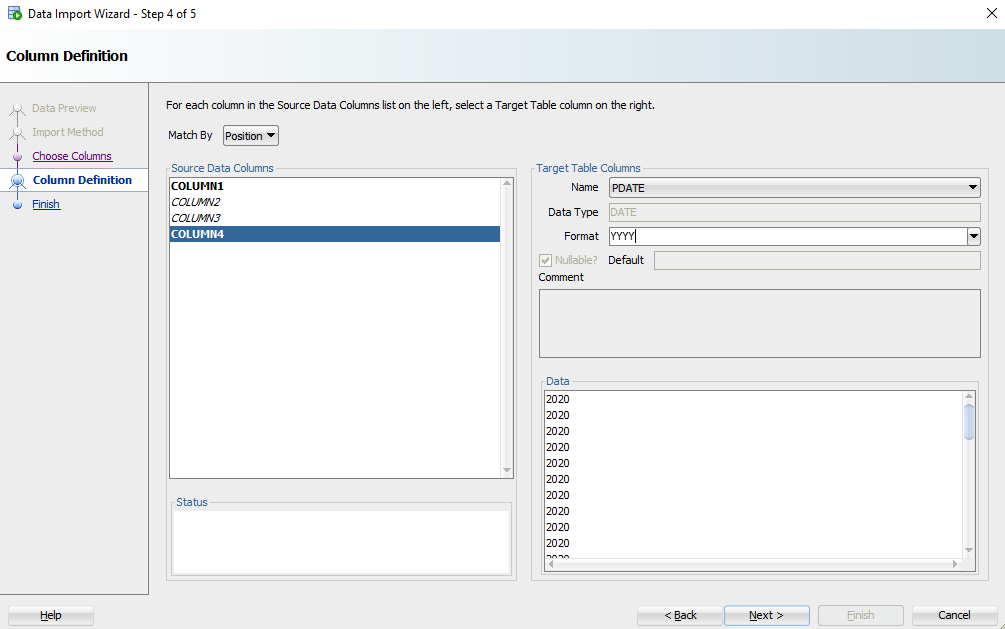
*foreign key(pid) references driver*

*)*

**Importing Dataset Into Oracle**



Choosing a file and making sure all data is separated.



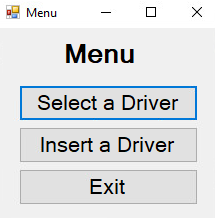
Changing any date formats.



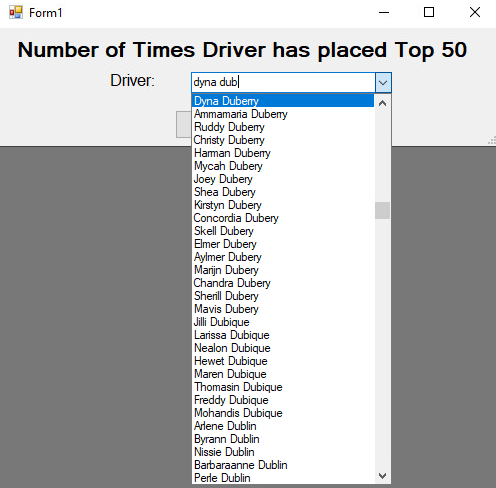
Data is shown to now be in the tables.

**Basic Application**

The application begins with a menu where we have the option to select a driver or to insert a driver. These are the two basic queries/inserts that we choose to implement into our program.

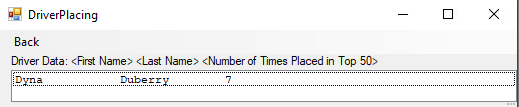
****

To start our application off, we decided to test a singular complex query. The first program allows users to see the number of times that a driver has placed in the top 50 of a Grand Prix race as this is the large database.

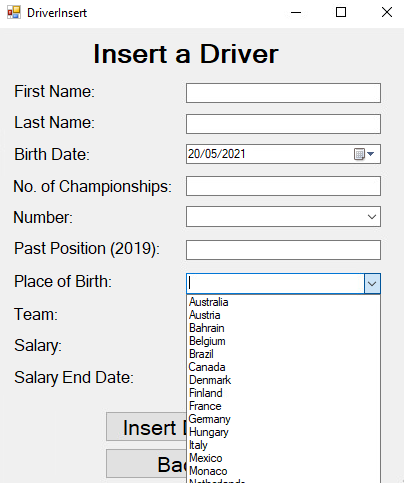


After you select a driver, you are directed to a new window which shows their id, first name , last name and the number of times they placed in the top 50.

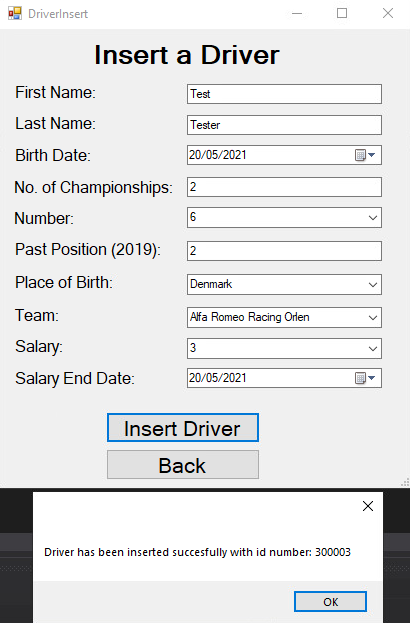
The program does not encounter errors as the user has to select a driver from the drop down.



If you select to insert a driver, the following interface appears. This gives the user all the necessary fields to enter information except for the id number which is automated. The birth date and salary end date use a date picker, while attributes like country and team have a drop down of all possibilities of the attribute depending on what is in the database.

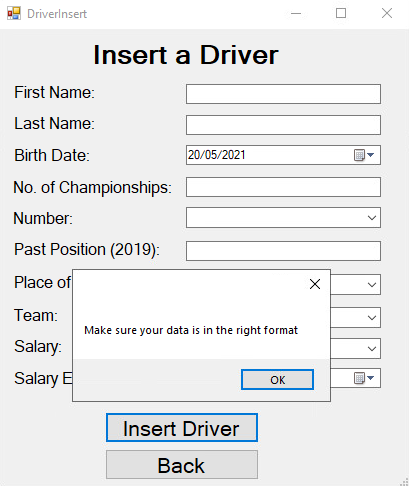
****

Here is shown if we insert a driver named ‘Test Tester’ with all attributes correctly entered, a message box appears saying it was correctly entered and says the id number of the driver entered. This id number is calculated by finding the highest id number of the drivers and adding one.

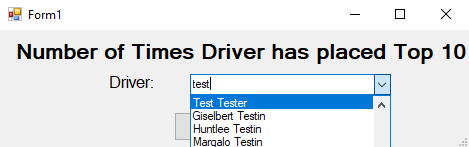
****

Note that if the user leaves the past position field blank, a driver will still be created as this is the only field which can be left blank.

Here if we enter no data a message box appears saying that you must check that your data is in the correct form. This also appears if the user enters a string instead of an integer.

****

We can now see under the driver select that the driver ‘Test Tester’ has appeared which means he was successfully entered into the database.

****

**Indexing**

To begin testing indexing, we decided to test both point and range queries. Here are the results from the queries before using indexing:

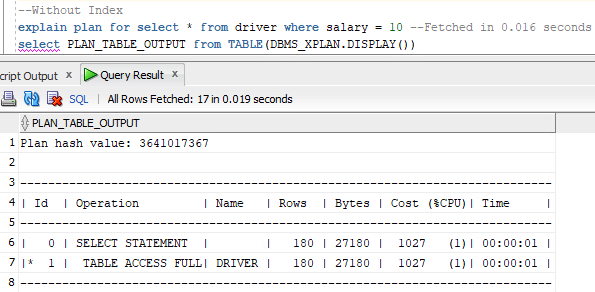
*select \* from driver where salary = 10 -- Fetched in 0.019 seconds*

*select \* from driver where salary > 990 and salary < 995 -- Fetched in 0.01 seconds*

*Select \* from driver where numChamp = 12 --Fetched in 0.016 seconds*

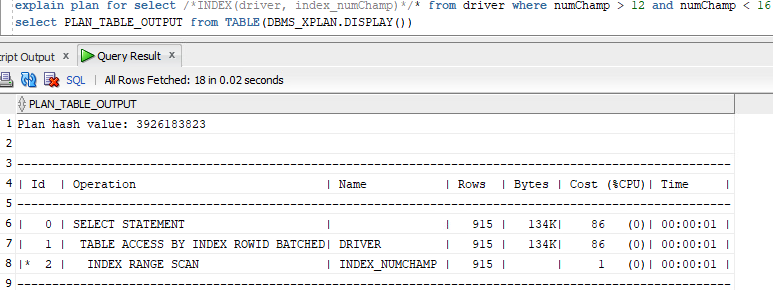
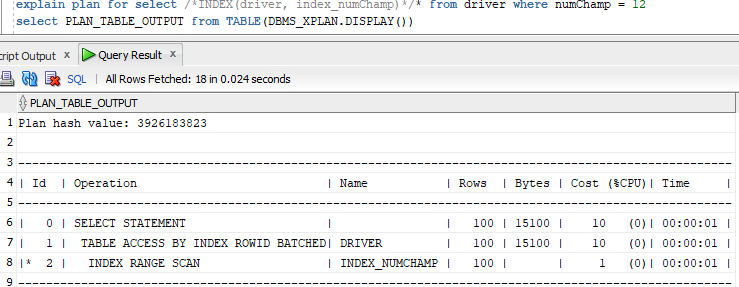
*Select \* from driver where numChamp > 12 and numChamp < 16 --Fetched in 0.01 seconds*

Without Index



As shown by the table above, line 1 has the operation table access full. This shows that Oracle is using a full table scan in order to find all drivers that have a salary that is equal to 10. This can be slow when you have a large amount of data, so indexing can be used to make it quicker.

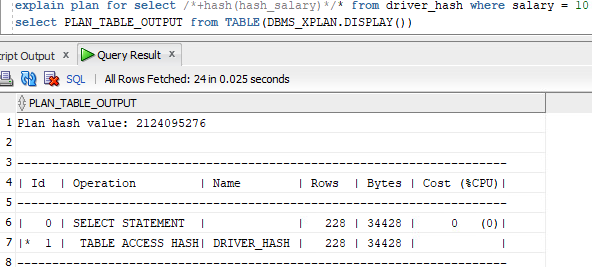
B-Tree Index

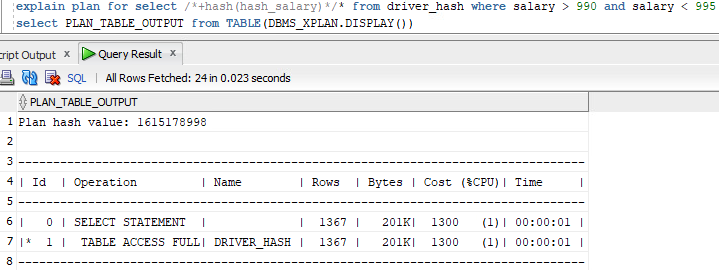


Here we have created a B-Tree index for the attribute that stores the number of championships for a driver (numChamp). The index has been used with a point query (numChamp=12) and a range query (numChamp > 12 and numChamp < 16). We have then analyzed the index using the explain plan feature of Oracle.

We are performing a range index scan using the index index\_numChamp. It has then accessed the table driver using the index rowIDs. Then it uses the select statement to filter through our data.

Hash Index





Here we have created a hash cluster for the attribute that stores the salary for a driver (salary). The index has been used with a point query (salary=10) and a range query (salary > 990 and salary < 995). We have then analyzed the cluster using the explain plan feature of Oracle.

We are performing a table access hash on the table driver\_hash. Driver\_hash was a temporary table created to be used for the hash cluster. Then it uses the select statement to filter through our data. As shown for the range query, Oracle actually used a table access full to search for the range of salaries as that is what Oracle believed to be the most cost efficient way to filter the attribute.

Comparing Indexes to Without Indexes

When comparing using the indexes to without indexes, the results are very similar. This is because computers are so fast these days that even with a large amount of data (300,000), the computer can easily perform a full table scan in less than a 1/10th of a second. However, when the data is millions of billions of rows, the indexes will help the computer perform better.

In theory, a hash index will perform best as a point query as we will only need to load in one bucket/block as the directory is kept in main memory. However, the hash index will perform worse than the B-tree index at a range query as 1 block is needed for each data point in the range.

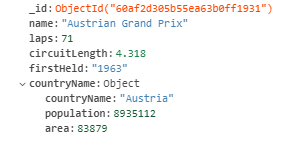
MongoDB Collections

When implementing our database into MongoDB, we decided to implement our collections as followed:

* One to Many Relationship: Create an object inside the collection which holds the data of the other object e.g Driver has an attribute countryName which holds a country object
* Many to Many Relationship: Created an array of objects in one of the collections that holds all objects e.g Driver has an attribute participates which is an array of objects



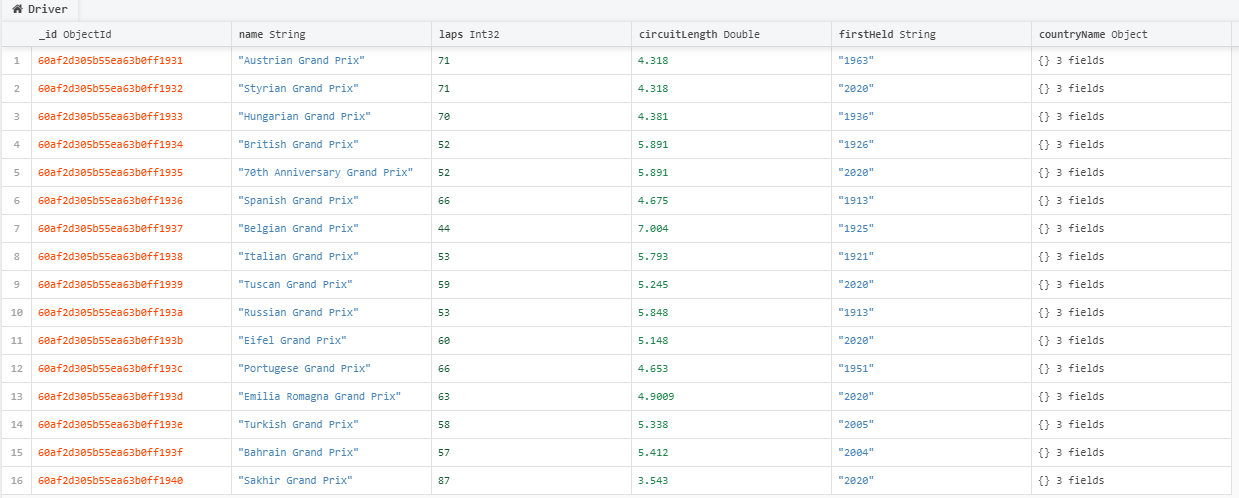
Above is the Driver collection. It has all the usual attributes where countryName is an object which holds the information about a country, teamName is an object which holds the information about a team and participates is an array which holds objects relating to the Driver participating in a Grand Prix. We decided to put participates inside the driver collection as in our program, the user will use the driver’s name to locate their participates relationship. We decided to do this as an array of objects as the dataset is quite small so it was easier to implement.



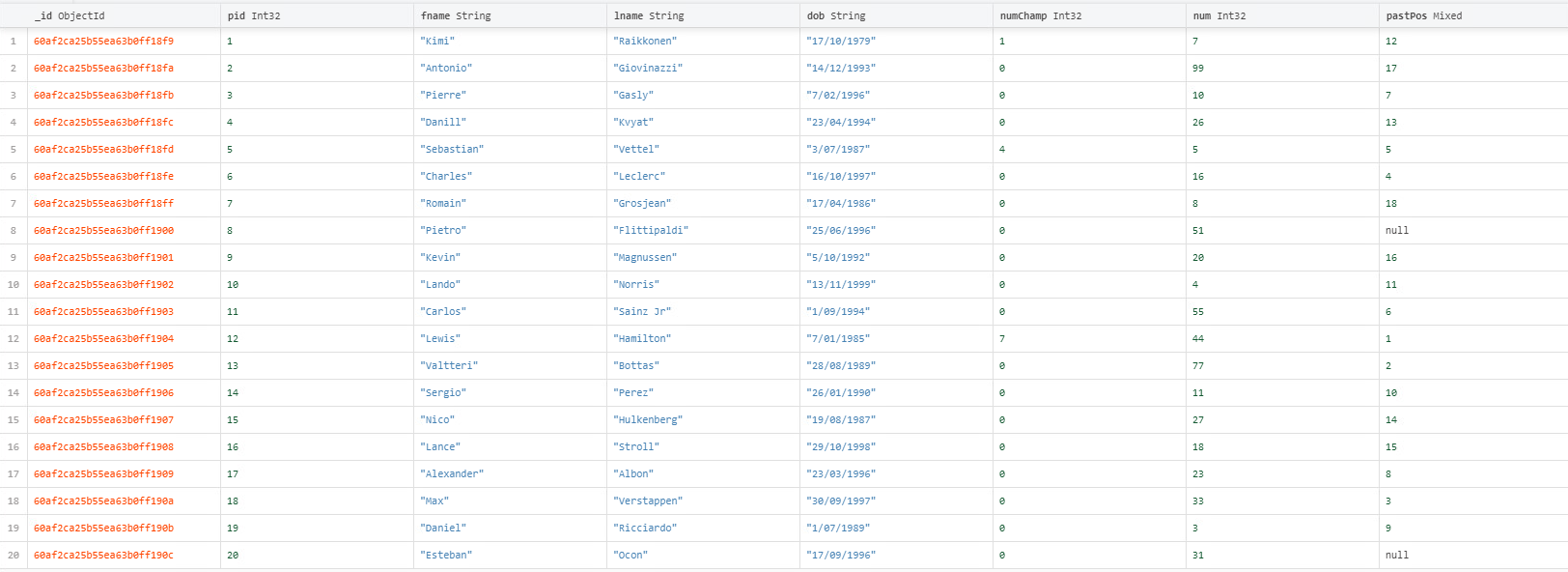
Above is the GrandPrix collection. It has all the usual attributes where countryName is an object which holds the information about a country.

Screenshot of the small dataset loaded into MongoDB.

GrandPrix Collection:

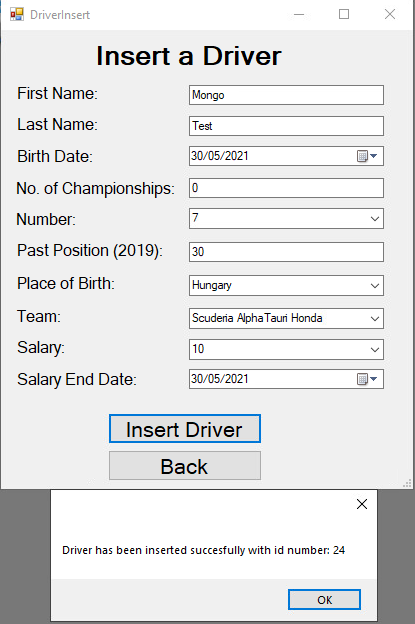
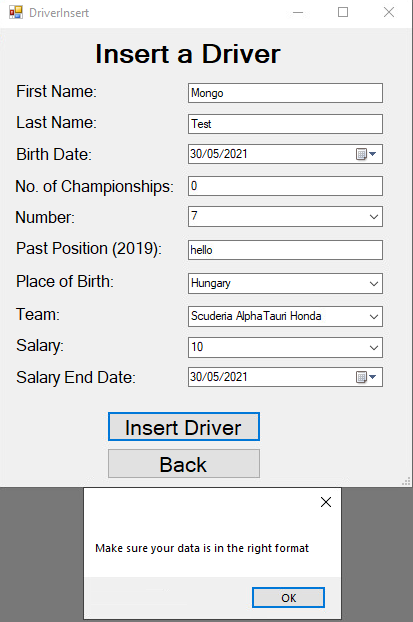


Driver Collection:



MongoDB and C# Program

This program has the same format as the previous program with Oracle SQL.

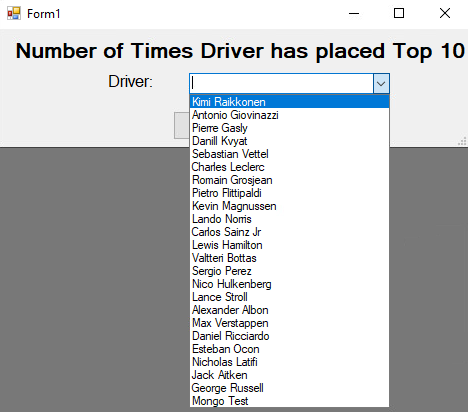


Note that if the user leaves the past position field blank, a driver will still be created as this is the only field which can be left blank.

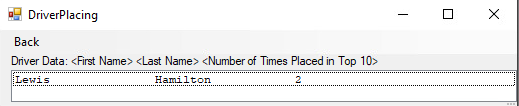
Program gives an error when an incorrect format is entered. This is because the user entered a string ‘hello’ in the past position textbox. When this is fixed the driver can be correctly entered. The id number is the next number in the sequence determined by the maximum id number in the driver collection.



As shown above, the driver Mongo Test contains all information about the country and team that the user inputted, including population, area and number of championships, even though they only selected the names from the dropdown.



You can see that at the bottom, the driver Mongo Test has been entered into the MongoDB database.



Can still select a driver to see how many times they have placed in the top 10. The only change that was made was we are now selecting top 10 places as the database is much smaller than the previous database.