**PRUEBA DE CONOCIMIENTO ALGORÍTMICO**

(programación orientada a objetos)

Duración de la prueba 4 horas

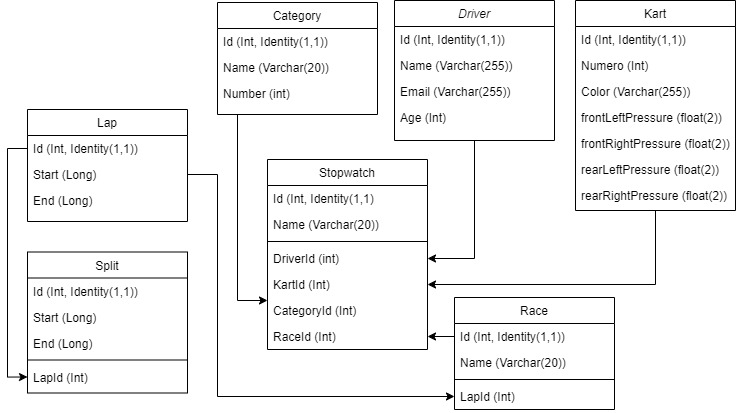
**Programa Cronometro**

**Nombre:**

**Cedula:**

El objetivo de la prueba lógica desarrollar la lógica de base de datos y algunos métodos vitales del sistema de Cronometro.

Considere el modelo se muestra a continuación:



**Detalle**

De cada Carrera (Race) se conoce:

* No existe un límite de carreras a cronometrar.
* Las carreras se pueden realizar de forma paralela.
* El sistema debe de generar un nombre automáticamente para cada carrera con un número es decir Race1, Race2 etc.
* Las Vueltas (Lap) deben estar asociadas a la carrera.
* Una carrera puede tener varias vueltas.
* Debe habar un total de duración de la carrera según el total general de los tiempos de las vueltas.

De cada vuelta (Lap) se conoce:

* Las vueltas deben de tener un tiempo final.
* En cada vuelta se puede realizar varios Cortes (Split) de tiempo.
* Las vueltas deben de tener un total general de tiempo siendo el resultado de la suma de los cortes (Split).
* Las vueltas se deben iniciar en 0:00:000.

De cada Corte (Split) se conoce:

* Los cortes deben de tener un tiempo final.
* Los cortes se deben iniciar en 0:00:000.

Del Cronometro (Stopwatch) se conoce:

* Debe permitir seleccionar al Conductor (Driver).
* Debe permitir seleccionar el vehículo (Kart) por el número del vehiculo.
* Debe permitir seleccionar la categoría, dependiendo de ella se aplicarán las siguientes reglas
  + Si es DD2 (Mayores de 25 años) los frenos serán en las 4 ruedas.
  + Si es DD2 Master (Mayores de 25 años experimentados) los frenos serán 4 ruedas.
  + Si es Junior (entre 18-25 años) los frenos serán en las 2 ruedas traseras.
  + Para DD2\* cuando es lluvia los frenos Delanteros 70% y los traseros 40% de frenado.
  + Para DD2\* si es terreno seco los frenos serán delanteros 0% traseros 100%.
* Debe mostrar en la pantalla la presión de las 4 ruedas
* Tiempo total de la carrera en tiempo.
* Tiempo total de la vuelta en tiempo.
* Las Vueltas con sus Split y los tiempos tanto de la vuelta como de los cortes.
* Debe permitir iniciar la carrera.
* Debe permitir detener la carrera.
* Debe permitir Marcar las vueltas.
* Debe permitir Marcar los Cortes de tiempo.
* Debe permitir guardar toda la información de la carrera según el modelo de datos no siendo este un obstáculo para el guardado de dicha información pudiéndose modificar según la necesidad.
* Debe permitir reiniciar la carrera con todos sus valores de tiempo en 0:00:000

1. Construya un mockup de la Interfaz del Cronometro comprendiendo todo lo antes estipulado en este documento.

<https://www.figma.com/design/E7XX5IE3aopUiAgnjsNWSX/StopWatch?node-id=0-1&t=tljaHjFL0e9U2Q3I-1>

1. Genere el script de base de datos partir del “modelo de datos propuesto” en este documento.

CREATE DATABASE StopWatch;

USE StopWatch;

-- Crear tabla Category

CREATE TABLE Category (

id INT IDENTITY(1,1),

categoryName VARCHAR(20),

number INT,

PRIMARY KEY(id)

);

-- Crear tabla Driver

CREATE TABLE Driver (

id INT IDENTITY(1,1),

driverName VARCHAR(255),

email VARCHAR(255),

age INT,

PRIMARY KEY(id)

);

-- Crear tabla Kart

CREATE TABLE Kart (

id INT IDENTITY(1,1),

numero INT,

color VARCHAR(255),

frontLeftPressure FLOAT(2),

frontRightPressure FLOAT(2),

rearLeftPressure FLOAT(2),

rearRightPressure FLOAT(2),

PRIMARY KEY(id)

);

-- Crear tabla Lap

CREATE TABLE Lap (

id INT IDENTITY(1,1),

lapStart BIGINT,

lapEnd BIGINT,

PRIMARY KEY(id)

);

-- Crear tabla Split

CREATE TABLE Split (

id INT IDENTITY(1,1),

splitStart BIGINT,

splitEnd BIGINT,

lapId INT,

PRIMARY KEY(id),

CONSTRAINT FK\_SplitLap

FOREIGN KEY (lapId) REFERENCES Lap(id)

);

-- Crear tabla Race

CREATE TABLE Race (

id INT IDENTITY(1,1),

raceName VARCHAR(20),

lapId INT,

PRIMARY KEY(id),

CONSTRAINT FK\_RaceLap

FOREIGN KEY (lapId) REFERENCES Lap(id)

);

-- Crear tabla Stopwatch

CREATE TABLE Stopwatch (

id INT IDENTITY(1,1),

stopwatchName VARCHAR(20),

driverId INT,

kartId INT,

categoryId INT,

raceId INT,

PRIMARY KEY(id),

CONSTRAINT FK\_StopwatchDriver

FOREIGN KEY (driverId) REFERENCES Driver(id),

CONSTRAINT FK\_StopwatchKart

FOREIGN KEY (kartId) REFERENCES Kart(id),

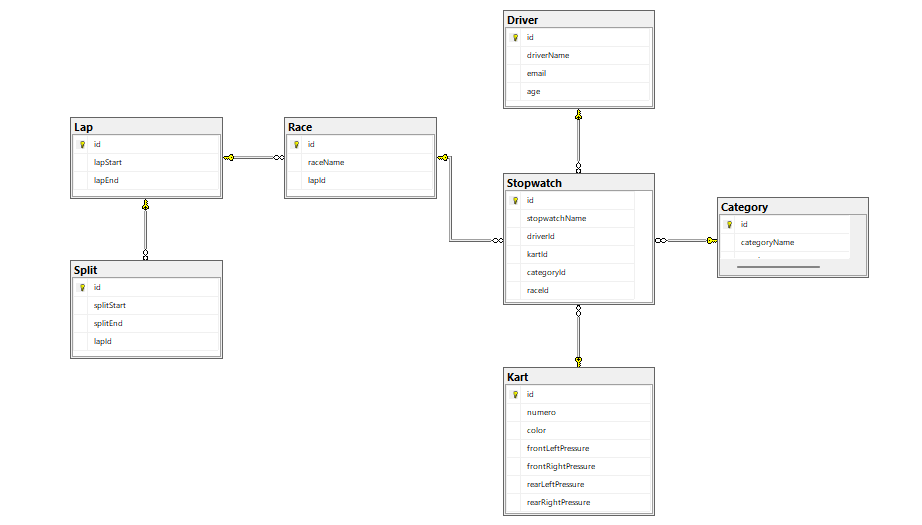
CONSTRAINT FK\_StopwatchCategory

FOREIGN KEY (categoryId) REFERENCES Category(id),

CONSTRAINT FK\_StopwatchRace

FOREIGN KEY (raceId) REFERENCES Race(id)

);



1. Genere el método en el lenguaje de su preferencia que según la categoría y el estado del terreno mostrara los tipos de frenos y el porcentaje utilizados en el vehículo para la carrera.
2. Genere en el lenguaje de su preferencia el método que guardara la información de la carrera sin conectarlo a una base de datos, solo la lógica del guardado.
3. Genere los métodos con la lógica de la carrera con las vueltas y los cortes de tiempo.
4. Genere el método para mostrar las categorías según la edad del conductor.

La implementación de los métodos solicitados en los puntos 3, 4, 5, 6 estara ubicados aquí: <https://github.com/dabog01/pruebaTecnica.git>

1. Mencione dos patrones de arquitectura de software y el caso de uso de la aplicación de estas.

**Patrón de Arquitectura MVC (Modelo-Vista-Controlador):**

**Modelo**: Aquí estaría toda la estructura de datos que maneja la información de las carreras, vueltas, cortes, conductores, karts y categorías. Es donde se calculan los tiempos totales, se manejan los nombres automáticos de las carreras y se guarda toda la información según cómo esté definido el modelo de datos.

**Vista**: Es la parte que ve el usuario, donde se muestra toda la información como los tiempos de vueltas, los cortes, la presión de las ruedas, etc. Esta vista sería interactiva, permitiendo iniciar, detener, marcar vueltas y cortes, reiniciar la carrera, etc.

**Controlador**: Este se encarga de gestionar las interacciones del usuario, como iniciar o detener la carrera, marcar vueltas y cortes, seleccionar conductor, kart y categoría, y actualizar la vista y el modelo según sea necesario.

**Caso de uso:** El MVC es útil porque separa claramente la parte lógica del negocio (modelo), la presentación (vista) y las interacciones del usuario (controlador). Esto ayuda a manejar la complejidad de las vueltas, cortes y reglas específicas de cada categoría, mientras se mantiene una interfaz de usuario interactiva y fácil de entender.

**Patrón de Arquitectura de Microservicios:**

**Microservicios**: Cada función del sistema (como manejar carreras, vueltas, conductores, etc.) podría ser un microservicio independiente. Cada uno tendría su propia lógica de negocio y base de datos, comunicándose entre sí mediante APIs bien definidas.

**Caso de uso:** Los microservicios permiten desarrollar, probar y desplegar cada función de manera independiente. Esto facilita la escalabilidad y mantenibilidad del sistema, ya que se pueden hacer ajustes y mejoras en cada componente sin afectar al resto del sistema.

1. Mencione brevemente 3 patrones de diseño y el caso de uso de la aplicación de estos.

**Singleton**

Garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a la misma.

**Casos de Uso**

- Cuando debe haber exactamente una instancia de una clase, y debe ser accesible desde un punto de acceso bien conocido.

- Cuando la instancia única debe ser extensible mediante subclases, y los clientes deben poder utilizar una instancia extendida sin modificar su código.

**Factory Method**

Define una interfaz para crear un objeto, pero permite a las subclases alterar el tipo de objetos que se crearán.

**Casos de Uso**

- Cuando una clase no puede anticipar la clase de objetos que necesita crear.

- Cuando una clase quiere que sus subclases especifiquen los objetos que crea.

- Para delegar la responsabilidad de crear instancias a las subclases.

**Observer**

Define una dependencia uno-a-muchos entre objetos de forma que cuando un objeto cambia de estado, todos sus dependientes son automáticamente notificados y actualizados.

**Casos de Uso**

- Cuando un cambio en un objeto requiere cambiar otros objetos.

- Para crear una relación uno a muchos entre objetos.

1. Mencione 2 metodologías de desarrollo.

**Scrum**

Es útil para trabajar en equipo teniendo en cuenta sus roles definidos (Scrum Master, Product Owner, Equipo de Desarrollo) y eventos (Daily Standup, Sprint Planning, Sprint Review, Sprint Retrospective) para gestionar el proyecto en un ambiente transparente y de constante retroalimentación, esta metodología ágil esta diseñada para tiempos cortos y entregables en cada sprint

**Kanban**

Es una metodología ágil que utiliza un tablero visual para gestionar el flujo de trabajo y mejorar la eficiencia del proceso. No tiene iteraciones fijas como Scrum, aun así, está en pro al trabajo continuo y la competitividad del equipo respecto a los entregables con sus estados Backlog, Stories, ToDo, InProgress, Testing, Done los cuales pueden tener algunas modificaciones según el proyecto

1. Mencione 1 un control de versión.

**Git**

Git permite gestionar los cambios en el código fuente y permite moverse a versiones anteriores del código, es útil para el trabajo en equipo ya que permite crear ramas de trabajo para enfoques específicos del proyecto y colaborar eficientemente a través de plataformas como GitHub y GitLab.

1. Realice las siguientes consultas de base de datos
   1. Seleccionar todos los conductores mayores de 25 años.
   2. Seleccionar los conductores en las diferentes categorías con mejor y peor tiempo de la carrera (Race).
   3. Seleccionar los conductores con menor tiempo en los cortes (Split).
   4. Seleccione el conductor con la vuelta más rápida (Lap).
   5. Seleccione los resultados del cronometro del día de la prueba.

-- Insertar datos en la tabla Category

INSERT INTO Category (categoryName, number)

VALUES ('DD2', 1),

('DD2 Master', 2),

('Junior', 3),

-- Insertar datos en la tabla Driver

INSERT INTO Driver (driverName, email, age)

VALUES ('Juan Perez', 'juan.perez@example.com', 30),

('Maria Gomez', 'maria.gomez@example.com', 22),

('Carlos Ruiz', 'carlos.ruiz@example.com', 28);

-- Insertar datos en la tabla Kart

INSERT INTO Kart (numero, color, frontLeftPressure, frontRightPressure, rearLeftPressure, rearRightPressure)

VALUES (1, 'Rojo', 32.5, 32.6, 32.7, 32.8),

(2, 'Azul', 32.4, 32.3, 32.6, 32.7),

(3, 'Verde', 32.6, 32.5, 32.8, 32.9);

-- Insertar datos en la tabla Lap

INSERT INTO Lap (lapStart, lapEnd)

VALUES (1634271600000, 1634271605000),

(1634271610000, 1634271615000),

(1634271620000, 1634271625000);

-- Insertar datos en la tabla Split

INSERT INTO Split (splitStart, splitEnd, lapId)

VALUES (1634271601000, 1634271602000, 1),

(1634271606000, 1634271607000, 1),

(1634271611000, 1634271612000, 2);

-- Insertar datos en la tabla Race

INSERT INTO Race (raceName, lapId)

VALUES ('Race1', 1),

('Race2', 2),

('Race3', 3);

-- Insertar datos en la tabla Stopwatch

INSERT INTO Stopwatch (stopwatchName, driverId, kartId, categoryId, raceId)

VALUES ('Stopwatch1', 1, 1, 1, 1),

('Stopwatch2', 2, 2, 3, 2),

('Stopwatch3', 3, 3, 1, 3);

-- a

SELECT \*

FROM Driver

WHERE age > 25;

-- b

SELECT

c.categoryName,

d.driverName,

MIN(l.lapEnd - l.lapStart) AS bestTime,

MAX(l.lapEnd - l.lapStart) AS worstTime

FROM

Stopwatch s

JOIN Driver d ON s.driverId = d.id

JOIN Category c ON s.categoryId = c.id

JOIN Race r ON s.raceId = r.id

JOIN Lap l ON r.lapId = l.id

GROUP BY

c.categoryName,

d.driverName;

--c

SELECT

d.driverName,

MIN(sp.splitEnd - sp.splitStart) AS minSplitTime

FROM

Stopwatch s

JOIN Driver d ON s.driverId = d.id

JOIN Lap l ON s.raceId = l.id

JOIN Split sp ON l.id = sp.lapId

GROUP BY

d.driverName

ORDER BY

minSplitTime;

--d

SELECT TOP 1

d.driverName,

l.lapEnd - l.lapStart AS lapTime

FROM

Stopwatch s

JOIN Driver d ON s.driverId = d.id

JOIN Race r ON s.raceId = r.id

JOIN Lap l ON r.lapId = l.id

ORDER BY

lapTime;

-- e

SELECT \*

FROM Stopwatch