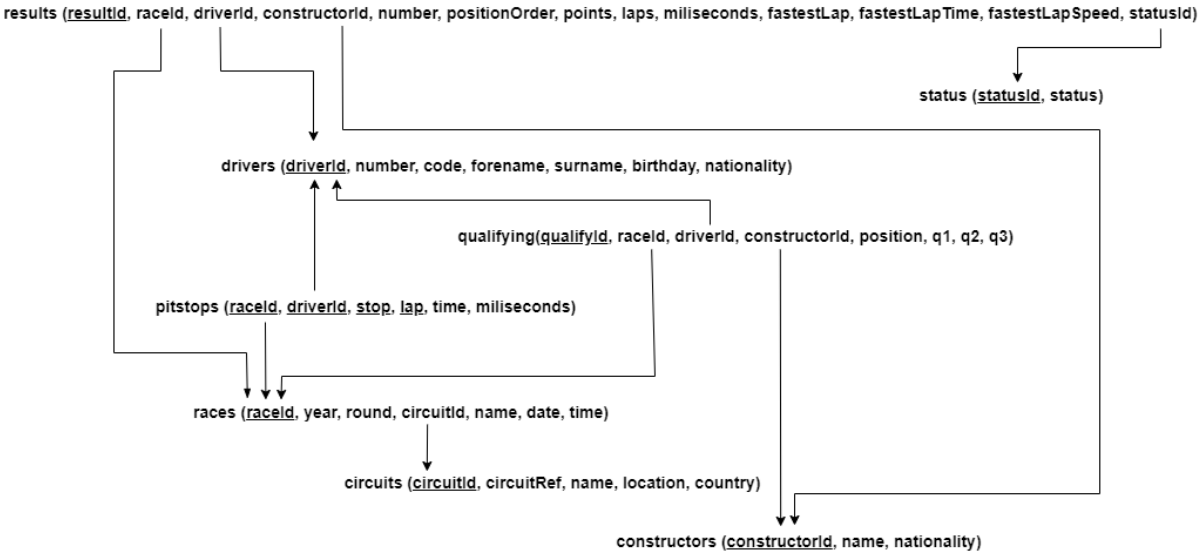
Bases de datos

Ejercicios de SQL

Carreras de Fórmula 1

La Federación Internacional de Automovilismo (FIA), como parte de su esfuerzo continuo para optimizar y mejorar la calidad de las carreras de Fórmula 1, ha tomado la decisión de recopilar y centralizar en una base de datos toda la información relevante relacionada con estos eventos de competición. Para ello, se ha diseñado una base de datos robusta y estructurada que consta de diversas tablas, las cuales tienen la función de almacenar de manera organizada toda la información referente a las carreras que se llevan a cabo durante los fines de semana en diferentes países del mundo. Estas tablas están diseñadas para capturar datos esenciales sobre cada carrera, como los equipos participantes, los pilotos, los resultados, las clasificaciones, y otros aspectos logísticos y técnicos del evento. La siguiente figura ilustra el modelo relacional de esta base de datos, proporcionando una visión clara de cómo se interconectan las diferentes tablas para ofrecer una representación integral de toda la información gestionada por la FIA en el contexto de la Fórmula 1.



A continuación se describe el contenido de las distintas tablas de la base de datos:

- `drivers`: Esta tabla almacena información sobre los pilotos que han participado en las carreras, incluyendo detalles como su nombre (`forename`), apellidos (`surname`), nacionalidad (`nationality`), y otros datos relevantes para su identificación.
- `results`: Contiene los resultados obtenidos por cada piloto en las carreras, registrando su posición final (`positionOrder`), puntos obtenidos (`points`) y cualquier otro dato que refleje su desempeño en la competición.
- `races`: En esta tabla se almacena toda la información pertinente a cada carrera, como el nombre del Gran Premio (`name`), la fecha en que se disputó (`date`) y el circuito en el que tuvo lugar.
- `circuits`: Incluye información detallada sobre los circuitos en los que se llevan a cabo las carreras, especificando su ubicación (`location`) y nombre (`name`) entre otros.
- `constructors`: Esta tabla registra a los constructores (equipos) que participan en las carreras. Generalmente, cada equipo compite con dos pilotos en cada carrera.
- `qualifying`: Almacena los datos sobre las sesiones de calificación que determinan el orden de salida de los pilotos antes de cada carrera. Normalmente, las rondas de calificación se disputan en tres tandas (`q1`, `q2` y `q3`), en las que los pilotos más lentos van siendo eliminados sucesivamente.
- `pitstops`: Registra las paradas que realizan los pilotos durante las carreras, ya sea para cambiar neumáticos, repostar combustible o realizar ajustes en el coche, detallando el momento (`lap`) y la duración de cada una (`miliseconds`).
- `status`: Esta tabla especifica el estado final de cada piloto al término de una carrera, indicando si finalizaron en condiciones normales, si perdieron vueltas, si tuvieron problemas técnicos o si abandonaron por alguna razón.

Adicionalmente, se hace constar que este modelo tiene añadidas las siguientes restricciones:

- Todos los identificadores se presentan de forma numérica.
- Los nombres, apellidos, nacionalidades y localizaciones no podrán exceder en ningún caso los 250 caracteres.
- Los campos que indican número de vuelta, puntos y velocidades tendrán que adaptarse como valores `INT`, `FLOAT` o `DOUBLE` según corresponda.
- Existen tres tipos adicionales de valores con formato temporal como son *time* de tipo `TIME`, *date* y *birthday* de tipo `DATE` y *year* de tipo `YEAR`.

Consultas SQL:

1. Obtener el nombre y apellidos de los pilotos españoles.
2. Obtener todos los datos de los circuitos alemanes.
3. Obtener los paises en los que se disputaron carreras en el año 2010.
4. Obtener el nombre de los pilotos que han participado en al menos 1 carrera del año 2016.
5. Nombre de los constructores con los que han disputado carreras más de 50 pilotos diferentes.
6. Nombre y apellidos de los pilotos que nunca han ganado una carrera.
7. Obtener el nombre y apellidos de los pilotos que durante el año 2017 han participado en todas las carreras.
8. Obtener el nombre, localización, país y año para cada circuito de las carreras que se han disputado entre 2015 y 2017, ordenado por el id del circuito.
9. Obtener los constructores que no han participado en alguna clasificación.
10. Obtener nombres, apellidos de los pilotos que han ganado más de 30 grandes premios así como el número de grandes premios que han ganado.
11. Nombre y apellidos del piloto que obtuvo la vuelta con velocidad media más alta así como el circuito y el año en el que se obtuvo.
12. Obtener el nombre, apellidos y la velocidad media del piloto que obtuvo la vuelta con velocidad media más alta en el gran premio de Japón de 2009.
13. Obtener el nombre de los pilotos que durante el año 2017 consiguieron puntos en todas las carreras.
14. Obtener el nombre de los pilotos, el circuito y el total de paradas, para aquellos pilotos que entre todos los grandes premios disputados han realizado en alguno de ellos el mayor número de paradas y también los que han realizado el menor número de ellas.
15. De entre todos los pilotos que han participado en todas las rondas de clasificación (Q1, Q2 y Q3) del gran premio de Abu Dhabi de 2017 (`qualifying.q1 <>'' AND qualifying.q2 <>'' AND qualifying.q3 <>''`), obtener el nombre de los pilotos y el id de los equipos, para aquellos equipos que tienen a sus dos pilotos en esa situación.
16. Obtener el nombre y apellido de los pilotos y el nombre de aquellas carreras en las que han participado pilotos rusos y polacos.
17. Obtener el nombre y apellidos de los pilotos y el número de vueltas totales recorridas en el año 2011 siempre y cuando sea mayor que la media del número de vueltas totales recorridas el año anterior por todos los pilotos.

18. Obtener el nombre y año de las carreras en las que se disputó una clasificación (`qualifying`) pero no se realizaron pitstops.
19. Obtener la nacionalidad de los pilotos que han disputado todas las ediciones del gran premio 'Australian Grand Prix'.
20. Eliminar de la tabla `qualifying` aquellas tuplas donde un piloto no haya participado en la clasificación.
21. Obtener aquellos constructores que habiendo ganado más de 5 carreras entre 2003 y 2010, no hayan participado en ninguna carrera desde el siguiente año.
22. Obtener el nombre de la carrera y el año en la que tuvo lugar todos los tipos de incidentes que se enumeran a continuación: descalificación, accidente, colisión, fallo de motor, caja de cambios y transmisión (`statusId` de 2 a 7).
23. Obtener nombre y apellidos del piloto, el nombre del circuito, el año de la carrera donde un piloto español obtuvo el tiempo de parada más pequeño (atributo `miliseconds`). Incluya este atributo en la salida de la consulta.
24. Codifique una consulta que obtenga el nombre de aquellos constructores que sean italianos (`nationality = Ítalian'`) con los que hayan disputado carreras al menos un piloto italiano.
25. Codifique una consulta que obtenga el nombre y apellidos del piloto que más accidentes (`status.status = 'Accident'`) ha tenido. Mostrar también el número de accidentes.
26. Codifique una consulta que obtenga el nombre y apellidos de los pilotos que hayan calificado entre los 10 primeros puestos (`position <= 10`) de todas las carreras del año 2015.
27. Obtener una lista con los nombres de aquellos constructores italianos ('Italian' en inglés) que nunca han competido con pilotos italianos.
28. Obtener toda la información de los constructores que en todas las carreras del año 2006, consiguieron que alguno de sus pilotos quedara entre los diez primeros de la clasificación.
29. Obtener nombre y apellidos del piloto que acabó la competición con más puntos entre los años 1990 y 2000, así como dicha suma de puntos.
30. Codifique una consulta que obtenga el nombre y apellidos de los pilotos que ganaron una carrera (`results.positionOrder = 1`) sin haber estado clasificados entre los 10 primeros pilotos (`qualifying.position >10`). Mostrar además el nombre de la carrera y el año en la que lo consiguieron.
31. Codifique una consulta que obtenga el nombre y apellidos del piloto que realizó más *pitstops* en una carrera del año 2013. Mostrar también el número de *pitstops*.
32. Codifique una consulta que obtenga el nombre y apellidos de los pilotos que hayan quedado entre los 10 primeros puestos (`positionOrder <= 10`) de todas las carreras del año 2017.

Procedimientos almacenados:

1. Crea un procedimiento almacenado de nombre `getRacesInAYear`, que obtenga como salida los nombres de las carreras y el número total de constructores que han participado en la carrera, para un año concreto que se pasará como parámetro de entrada. En el procedimiento no se definirán parámetros de salida.
2. Se quiere testear los mensajes que se le envían a los pilotos por pantalla en plena carrera. Para ello se quiere crear un procedimiento de nombre `getOnRaceMessages`, que reciba un código de mensaje, e indique en un parámetro de salida el mensaje concreto teniendo en cuenta las siguientes condiciones:
   - E01 = Error en la presión de las ruedas

- E02 = Pinchazo
- E03 = Temperatura alta en el motor
- E04 = Frenos sobre-calentados
- E05 = Error presión del aceite
- Cualquier otro valor = Error de comando

3. Crea un procedimiento almacenado para obtener los pilotos y los circuitos que ganaron carreras de un año concreto (como argumento) con un constructor del mismo país que el piloto.
4. Escribe un procedimiento que realice la selección de aquellos pilotos que en el año que se le pase como parámetro de entrada, obtuvieron un primer, un segundo y un tercer puesto.
5. Procedimiento almacenado de nombre `getsConstructoresYPilotos`, que tomando como entrada un año concreto, obtenga como salida los nombres de todos los pilotos que participaron en el mundial ese año y el equipo de constructores con el cual compitieron. Adicionalmente, también se quiere obtener el total de puntos que obtuvo cada piloto en el mundial.
6. Codifique un procedimiento almacenado denominado `getNumberOfVictories` que disponga de un parámetro de entrada `type`. Si `type = ñationality'`, el procedimiento listará el número de victorias que han obtenido cada una de las nacionalidades de la tabla `drivers`. Por contra, si `type` toma cualquier otro valor, el procedimiento listará el número de victorias que han obtenidos cada uno de los constructores (`constructors.name`) existentes en la base de datos. Los resultados deberá estar ordenados de mayor a menor número de victorias.
7. Añade una nueva columna en la tabla `drivers` que se llame años en activo. A continuación escribir un procedimiento que, haciendo uso de un `Cursor`, calcule el número total de años que ha estado un piloto compitiendo y actualice la tabla.
8. Crea un procedimiento que reciba una nacionalidad como parámetro de entrada y realice una consulta sobre la tabla `drivers` para obtener todos los pilotos de esa nacionalidad. Los nombres y apellidos de estos pilotos deberán devolverse en un parámetro de salida de tipo `TEXT` separados por comas (`,`).

Funciones almacenadas:

1. Escribe una función que devuelva el número de puntos que se ha conseguido el campeón de ese año en cada mundial.
2. Escribe una función que devuelva el valor medio de puntos por año que ha conseguido un determinado constructor que se recibirá como parámetro de entrada. El parámetro de entrada será el nombre del constructor.
3. Escribe una función que dado el `driverId` de un piloto devuelva el número total de años en activo que ha estado compitiendo.
4. Codifique una función almacenada llamada `diffPoints` que reciba los identificadores de dos pilotos como parámetros (`driver1` y `driver2`) y devuelva la diferencia de puntos totales conseguidos por `driver1` con respecto de `driver2`. Tenga en cuenta que el valor será positivo si `driver1` ha conseguido más puntos que `driver2` y negativo en caso contrario.

Triggers:

1. Desarrolle taodos los triggers necesarios para impedir que en una misma carrera no puedan participar más de 2 pilotos con un constructor. Se considera que participar en una carrera es aparecer en la tabla `results` (se ignora por tanto la tabla `qualifying`).

2. Crea una tabla `crashes` con la siguiente estructura:

crashId	driverId	descripción

   Crea un trigger que cuando se inserte una fila en la tabla `results` con el parámetro `statusId` de valores 3 ó 4 rellene la tabla `crashes` con la información correspondiente.

3. Crea un trigger para impedir que en un mismo año un piloto participe en carreras con dos constructores diferentes. Se considera que un piloto habrá participado en una carrera si aparece en la tabla `results`.

4. Cree una nueva tabla *sponsors* en la base de datos que almacene como atributos, su identificador propio, su nombre, tipo, el identificador de la carrera que patrocina (un patrocinador solo puede patrocinar una carrera) y el dinero que aporta anualmente. Existen dos tipos de patrocinadores: oficiales, deben aportan una cantidad igual o superior a 5M de euros, y cooficiales, para cantidades inferiores. Diseñe un trigger que clasifique al sponsor y rellene el atributo *type* de su correspondiente tabla.

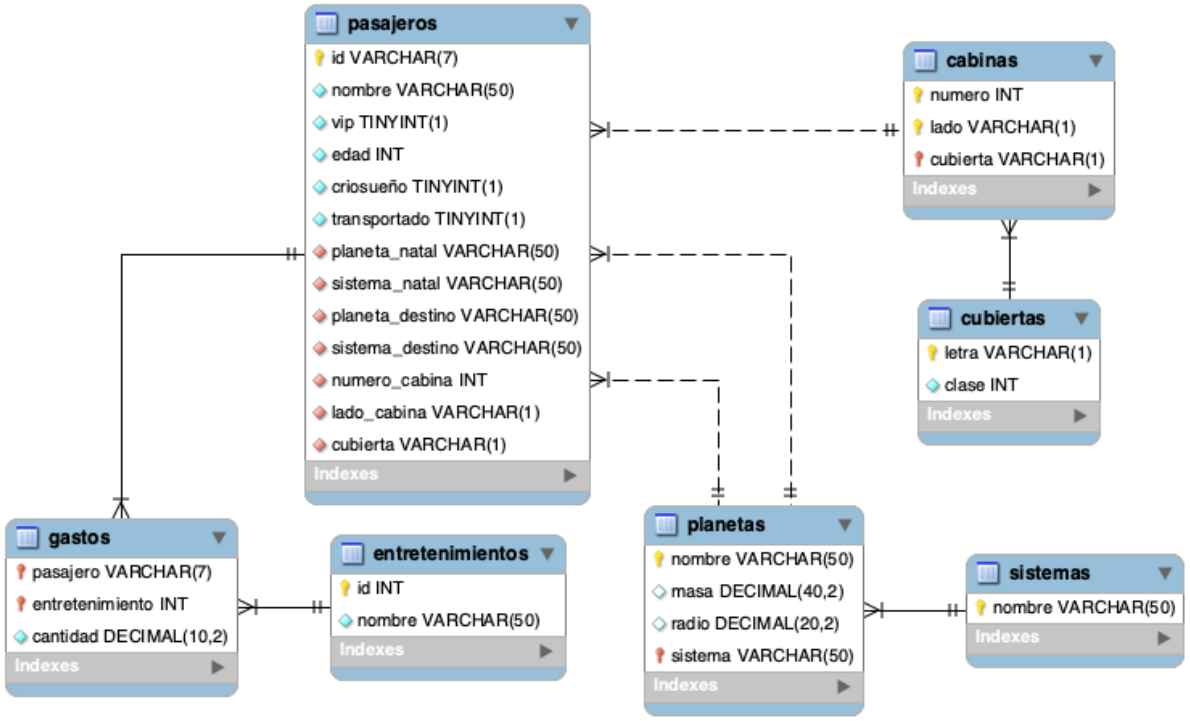
Titanic Spaceship

En el año 2912, la humanidad ha ampliado sus horizontes más allá de la Tierra, estableciendo colonias en diversos planetas y lunas del universo. Las ciudades espaciales flotantes se han transformado en centros vibrantes de investigación y comercio, donde la vida cotidiana se entrelaza con tecnología avanzada y una rica diversidad cultural. Viajar entre planetas se ha vuelto algo habitual, y la exploración de nuevos mundos se considera una prioridad, aunque no sin desafíos. A pesar de los avances, persisten tensiones sociales y desigualdades, ya que algunos disfrutan de las comodidades del espacio mientras otros enfrentan la dura realidad de la vida en las periferias. Sin embargo, cada nuevo descubrimiento alimenta la esperanza de un futuro más unido y sostenible para la humanidad.

En la sede de la Agencia Espacial Europea, se ha recibido un mensaje alarmante de una fuente situada a cuatro años luz de distancia. La comunicación revela una situación preocupante relacionada con la Nave Espacial Titanic, un transatlántico interestelar que partió hace un mes con más de 8.000 pasajeros a bordo. La nave se encontraba en su viaje inaugural, transportando emigrantes de nuestro sistema solar hacia tres exoplanetas recién habitables que orbitan en estrellas cercanas.

Mientras rodeaba Alfa Centauri en ruta a su primer destino, el cálido 55 Cancri E, la Nave Espacial Titanic colisionó con una anomalía del espacio-tiempo oculta dentro de una nube de polvo. Aunque la nave permaneció intacta, casi la mitad de los pasajeros fueron transportados a una dimensión alterna.

Para ayudar a los equipos de rescate y recuperar a los pasajeros perdidos, es crucial ralizar un análisis de la base de datos que contiene información detallada sobre cada pasajero. Tras grandes esfuerzos por parte de los mejores ingenieros informáticos con los que Agencia Espacial Europea ha podido contar, se ha recuperado el siguiente esquema de la base de datos:



Resuelve los siguientes ejercicios para poder traer de vuelta a esta dimensión a los pasajeros perdidos.

Consultas SQL

1. Obtener el listado de los pasajeros que han embarcado obteniendo para cada uno su nombre, cabina asignada (con formato: Letra Cubierta-Número Cabina-Lado Cabina, por ejemplo “B-1-P”) y planeta destino al que viajaban (incluyendo el sistema al que pertenece), de aquellos pasajeros cuya cabina esta localizada en una cubierta de tercera clase desde la letra A a la D y que no están en criosueño.

2. Obtener para cada zona de entretenimiento el gasto total que han tenido los pasajeros VIP en ella

mostrando el nombre de la zona, el gasto total y el tipo de pasajero (“VIP”), haciendo lo mismo con el gasto correspondiente a los pasajeros que no sean VIP para los cuales se mostrará el valor “NO VIP” en el tipo de pasajero. Tipo de pasajero se mostrará como una columna que podrá tomar los dos posibles valores mencionados anteriormente (“VIP” o “NO VIP”).

3. Obtener el nombre de los planetas, y el sistema al que pertenecen, de aquellos planetas en los que han nacido menos pasajeros que el promedio de nacimientos en todos los planetas.
4. Obtener el nombre de los pasajeros y su gasto total por zona de entretenimiento de aquellos pasajeros alojados en la cubierta ‘A’y que han realizado gastos en todas los zonas de entretenimiento, ordenados por nombre de pasajero en orden alfabético y por cada pasajero que aparezcan los gastos de mayor importe primero.
5. Obtener las cabinas (mostrando la información de cubierta, número y lado) ocupadas por pasajeros que han gastado en el entretenimiento más popular, es decir en aquel en el que han participado más pasajeros.
6. Obtener los nombres de los planetas, así como el sistema al que pertenecen, de aquellos planetas a los que no han viajado pasajeros VIP y que además son los planetas donde han nacido alguno de los pasajeros con mayor edad.
7. Obtener aquellas cabinas (mostrando la información de cubierta, número y lado) con más de 3 pasajeros NO VIP y cuyos pasajeros han gastado más del doble del gasto promedio total por cabina.
8. Obtener los planetas donde la cantidad de pasajeros nacidos en él, y que han gastado en al menos 4 tipos diferentes de entretenimiento, es mayor al 25 % del total de pasajeros de dicho planeta, mostrando además del nombre del planeta y sistema al que pertenece, tanto el total de pasajeros nacidos en él, así como el número de pasajeros del mismo que gastaron en los tipos de entretenimiento.

Vistas:

1. Crea una nueva vista denominada entretenimiento_pasajeros que contenga los siguientes datos:
   - id, nombre, y edad de cada pasajero.
   - Si el pasajero fue o no transportado.
   - nombre y sistema del planeta al que está viajando el pasajero.
   - numero, lado y cubierta de la cabina en la que está hospedado y la clase de la cubierta asociada.
   - nombre del entretenimiento en el que más ha gastado el pasajero, junto con la cantidad gastada en dicho entretenimiento.

Procedimientos almacenados:

1. Para facilitar una identificación más rápida y precisa de los tutores legales de los menores en cada cabina, se ha determinado la necesidad de añadir un parámetro adicional en el registro de cada pasajero. Este parámetro permitirá identificar al tutor legal en caso de que el pasajero sea menor de edad. Para implementar este cambio, es necesario actualizar la tabla pasajeros y añadir un nuevo campo denominado tutor.

   Tras modificar la estructura de la tabla, se desarrollará un procedimiento para asignar automáticamente el tutor de cada menor. Este procedimiento tomará en cuenta que el tutor de cada pasajero menor de 18 años debe ser un adulto de mayor edad que se encuentre en la misma cabina (con la misma letra, número y lado) que el menor. Si no hubiera un adulto en la misma cabina que pueda cumplir esta función, el menor deberá colocarse en criosueño, salvo que ya se encuentre en esta condición. Los menores que ya estén en criosueño no requerirán asignación de tutor, permaneciendo en esta condición sin cambios adicionales.

Funciones almacenados:

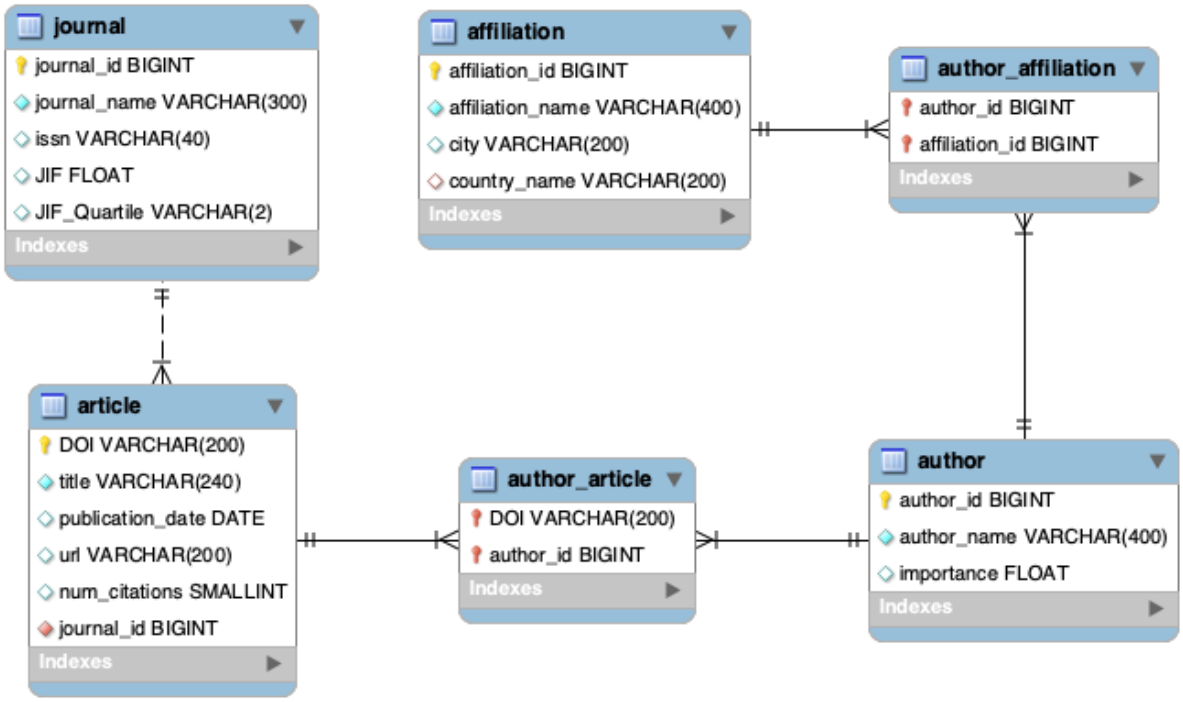
1. Se quiere crear una función que permita obtener el porcentaje de pasajeros que viajan a un destino concreto (sistema y planeta), para una cubierta determinada, cuyos datos pasaremos como parámetro. A fin de comprobar su correcto funcionamiento, crear una consulta que muestre la ocupación de todas las cubiertas disponibles para el destino “55 Cancri e” sistema “Copernico” ordenada alfabéticamente por la letra de la cubierta.

Triggers:

1. Para evitar que pueda haber algún ciber-delincuente en alguno de los entretenimientos ofrecidos por en la nave, se quieren crear tantos triggers como sean necesarios para evitar que existan gastos en entretenimientos con saldo negativo, es decir, la cantidad en gastos sea menor que 0. En caso de producirse el ciber-ataque, deberá mostraserse el siguiente mensaje de error: “*ERROR: Bienvenido ciber-delincuente, pero no puedes sacar dinero en un entretenimiento*”.
2. Para mejorar la satisfacción de los pasajeros, se ha decidido asignar automáticamente la condición de VIP a cualquier pasajero cuyo gasto total en entretenimientos supere la cantidad de 5000. Actualice la condición de VIP a todos los pasajeros que ya hayan superado los 5000 en gastos mediante una sentencia `UPDATE` y, además, cree los triggers necesarios para otorgar dicha condición a futuros pasajeros.

Artículos científicos

Un conjunto de editoriales europeas busca automatizar la gestión de sus revistas, autores y artículos mediante una base de datos que contemple sus características esenciales. Las revistas se clasifican por su factor de impacto y cuartil, y se publican periódicamente bajo una editorial. Los artículos, identificados por su DOI, título, fecha de publicación, URL y citas, se asocian a una revista y a uno o varios autores, quienes también tienen afiliaciones institucionales y pueden pertenecer a grupos de investigación vinculados a un área específica. El sistema debe registrar procesos editoriales como el envío a revisores, sus evaluaciones y fechas, así como colaboraciones informales entre autores y referencias entre artículos de la propia base. El siguiente diagrama contiene el esquema de la base de datos que guarda esta información.



Consultas SQL:

1. Obtener los nombres, en orden alfabético, de aquellos autores que perteneciendo a la “Universidad Politécnica de Madrid” han publicado algún artículo en el año 2020 o en el año 2021.
2. Obtener los nombres, en orden alfabético, de aquellos autores que perteneciendo a la “Universidad Politécnica de Madrid” han publicado algún artículo en el año 2020 y también en el año 2021.
3. Obtener el nombre de los autores y el nombre de la afiliación de aquellos autores que, perteneciendo a alguna entidad española, no han publicado ningún artículo ni en 2020 ni en 2021, ordenados por afiliación y dentro de cada entidad, por nombre de autor.
4. Obtener el nombre de la revista, su issn y el total de citas (`num_citations`) de todos los artículos publicados para cada una de ellas en aquellas revistas que están clasificadas dentro del primer cuartil de factor de impacto (`q1`).
5. Obtener el nombre de la revista y el total de citas (`num_citations`) que hayan recibido sus artículos para aquella/s revista/s que, perteneciendo al primer cuartil del factor de impacto (`q1`), tengan el mayor número de citas de toda la base de datos.
6. Obtener aquellas entidades (`affiliation_name`) que tienen asociados al menos 10 autores que hayan publicado más de 5 artículos en el año más reciente que figuren en la base de datos (este año debe calcularse de forma dinámica con la consulta).

7. Obtener el nombre de aquellas revistas que, habiendo publicado más de 300 artículos en el año de mayor antigüedad que figure en la base de datos tengan un factor de impacto (`jif`) superior a la media de los factores de impacto del global de las revistas de la base de datos. El año debe calcularse de forma dinámica con la consulta.
8. Obtener el nombre de aquellas revistas que hayan recibido en el global de sus artículos un mayor número de citas que la media de las citas recibidas por cada revista para el global de sus artículos de la base de datos.
9. Obtener el nombre de aquellos autores que hayan publicado artículos en todos los diferentes años que figuren en la base de datos.

Procedimientos alamacenados:

1. Crear un procedimiento que, recibiendo un año como parámetro, devuelva, en dos parámetros de salida, el nombre de la revista y el número de autores para aquella revista que haya publicado en dicho año el artículo con un mayor número de autores. Si existen varias revistas con el máximo de autores, habrá que proporcionar como resultado una cadena que contenga, separados por puntos y comas, los nombres de dichas revistas (en un parámetro de salida solamente puede devolverse un valor). Se deberá usar obligatoriamente, al menos, un cursor en la resolución del procedimiento.

Funciones alamacenadas:

1. Crear una función que, recibiendo como parámetro un identificador de una revista devuelva el número medio de artículos por año que dicha revista ha publicado. Escribir el código necesario para poder probarlo.

Triggers:

1. Crear la tabla correspondiente a la asignación y realización de las revisiones de los artículos por parte de los autores (que actúan como revisores) que se ha obtenido en el análisis inicial. Una vez creada, crear un *trigger* que impida que se pueda insertar a un revisor de un artículo si dicho revisor figura ya como autor del mismo. Probar su funcionamiento con el código necesario.

Soluciones

Las soluciones que se presentan a continuación pueden diferir de las discutidas en clase, ya que están basadas en las decisiones y criterios aplicados por el profesor al momento de su elaboración. Es importante señalar que estas soluciones podrían contener errores tipográficos o imprecisiones que no afectan su propósito principal.

Carreras de Fórmula 1

Consultas SQL

1. ```sql
   SELECT surname, forename
   FROM drivers
   WHERE nationality = 'Spanish';
   ```

2. ```sql
   SELECT *
   FROM circuits
   WHERE country = 'Germany';
   ```

3. ```sql
   SELECT DISTINCT country
   FROM circuits
       INNER JOIN races ON races.circuitId = circuits.circuitId
   WHERE year = 2010
   ```

4. ```sql
   SELECT DISTINCT drivers.surname, drivers.forename
   FROM drivers
       INNER JOIN results ON drivers.driverId = results.driverId
       INNER JOIN races ON results.raceId = races.raceId
   WHERE year = 2016
   ```

5. ```sql
   SELECT name
   FROM constructors
   WHERE constructorId IN (SELECT constructorId
                           FROM results
                           GROUP BY constructorId
                           HAVING COUNT(DISTINCT driverId) > 50)
   ```

6. ```sql
   SELECT forename, surname
   FROM drivers
   WHERE driverId NOT IN (SELECT driverId
                   FROM results
                   WHERE positionOrder = 1)
   ```

7. ```sql
   SELECT drivers.surname, drivers.forename
   FROM drivers
   WHERE driverId IN (SELECT driverId
                      FROM results
                      WHERE raceId IN (SELECT raceId FROM races WHERE year=2017)
                      GROUP BY results.driverId
                      HAVING COUNT(distinct results.raceId) = (SELECT COUNT(*)
                                                               FROM races
                                                               WHERE year=2017))
   ```

8. ```sql
   SELECT circuits.circuitId, circuits.name, circuits.location, circuits.country, year
   FROM races
       INNER JOIN circuits ON races.circuitId = circuits.circuitId
   WHERE year BETWEEN 2015 AND 2017
   ORDER BY circuits.circuitId
   ```

9. ```sql
   SELECT *
   FROM constructors
   WHERE constructors.constructorId NOT IN (SELECT qualifying.constructorId
                                            FROM qualifying)
   ```

10. ```sql
    SELECT drivers.surname, drivers.forename, COUNT(*) AS wins
    FROM drivers
        INNER JOIN results ON results.driverId = drivers.driverId
    WHERE positionOrder = 1
    GROUP BY drivers.driverId, drivers.surname, drivers.forename
    HAVING COUNT(*) >= 30
    ```

11.
```
SELECT drivers.forename, drivers.surname, circuits.name, races.year, results.
    fastestLapSpeed
FROM drivers
    INNER JOIN results ON drivers.driverId = results.driverId
    INNER JOIN races ON races.raceId = results.raceId
    INNER JOIN circuits ON circuits.circuitId = races.circuitId
WHERE fastestLapSpeed >= ALL (SELECT fastestLapSpeed
                              FROM results)
```

12.
```
SELECT drivers.forename, drivers.surname, results.fastestLapSpeed
FROM drivers
    INNER JOIN results ON results.driverId = drivers.driverId
    INNER JOIN races ON races.raceId = results.raceId
WHERE races.year = 2009
    AND races.name = 'Japanese Grand Prix'
    AND results.fastestLapSpeed >= ALL (SELECT fastestLapSpeed
                                        FROM results
                                        INNER JOIN races ON races.raceId = results.
                                            raceId
                                        WHERE year = 2009
                                        AND name = 'Japanese Grand Prix')
```

13.
```
SELECT drivers.forename, drivers.surname
FROM results
    INNER JOIN races ON races.raceId = results.raceId
    INNER JOIN drivers ON drivers.driverId = results.driverId
WHERE points > 0
    AND year = 2017
GROUP BY drivers.driverId, drivers.forename, drivers.surname
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM races
                   WHERE year = 2017)
```

14.
```
SELECT drivers.forename, drivers.surname, circuits.name, races.year, COUNT(*)
FROM drivers
    INNER JOIN pitstops ON pitstops.driverId = drivers.driverId
    INNER JOIN races ON races.raceId = pitstops.raceId
    INNER JOIN circuits ON circuits.circuitId = races.circuitId
GROUP BY drivers.driverId, drivers.forename, drivers.surname, races.raceId, races.
    year, circuits.name
HAVING COUNT(*) >= ALL(SELECT COUNT(*)
                       FROM pitstops
                       GROUP BY driverId, raceId)
```

15.
```
SELECT drivers.forename, drivers.surname, constructors.name
FROM drivers
    INNER JOIN qualifying ON qualifying.driverId = drivers.driverId
    INNER JOIN constructors ON qualifying.constructorId = constructors.constructorId
    INNER JOIN races ON races.raceId = qualifying.raceId
WHERE races.year = 2017
    AND races.name = 'Abu Dhabi Grand Prix'
    AND constructors.constructorId IN (SELECT qualifying.constructorId
                                       FROM qualifying
                                           INNER JOIN races ON races.raceId =
                                               qualifying.raceId
                                       WHERE races.year = 2017
                                         AND races.name = 'Abu Dhabi Grand Prix'
                                         AND qualifying.q1 <> ''
                                         AND qualifying.q2 <> ''
                                         AND qualifying.q3 <> ''
                                       GROUP BY qualifying.constructorId
                                       HAVING COUNT(DISTINCT qualifying.driverId) =
                                           2)
ORDER BY constructors.name ASC
```

16.
```
SELECT drivers.forename, drivers.surname, races.name, races.year
FROM drivers
    INNER JOIN results ON results.driverId = drivers.driverId
    INNER JOIN races ON races.raceId = results.raceId
```

```
WHERE races.raceId IN (SELECT raceId
                       FROM results
                           INNER JOIN drivers ON drivers.driverId = results.driverId
                       WHERE drivers.nationality = 'Russian')
  AND races.raceId IN (SELECT raceId
                       FROM results
                           INNER JOIN drivers ON drivers.driverId = results.driverId
                       WHERE drivers.nationality = 'Polish')
```

17.
```
SELECT drivers.forename, drivers.surname, SUM(results.laps)
FROM drivers
    INNER JOIN results ON results.driverId = drivers.driverId
    INNER JOIN races ON races.raceId = results.raceId
WHERE races.year = 2011
GROUP BY drivers.driverId, drivers.forename, drivers.surname
HAVING SUM(results.laps) > (SELECT AVG(nLaps)
                            FROM (SELECT SUM(laps) AS nLaps
                                  FROM results
                                      INNER JOIN races ON races.raceId = results.
                                          raceId
                                  WHERE year = 2010
                                  GROUP BY driverId) t)
```

18.
```
SELECT name, year
FROM races
WHERE raceId IN (SELECT raceId FROM qualifying)
  AND raceId NOT IN (SELECT raceId FROM pitstops)
```

19.
```
SELECT drivers.nationality
FROM drivers
    INNER JOIN results ON results.driverId = drivers.driverId
    INNER JOIN races ON races.raceId = results.raceId
WHERE races.name = 'Australian Grand Prix'
GROUP BY drivers.driverId, drivers.nationality
HAVING COUNT(*) = (SELECT COUNT(*)
                   FROM races
                   WHERE name = 'Australian Grand Prix')
```

20.
```
DELETE FROM qualifying
WHERE q1 = '' AND q2 = '' AND q3 = ''
```

21.
```
SELECT *
FROM constructors
WHERE constructorId IN (SELECT results.constructorId
                        FROM results
                            INNER JOIN races ON races.raceId = results.raceId
                        WHERE races.year BETWEEN 2003 AND 2010
                          AND results.positionOrder = 1
                        GROUP BY results.constructorId
                        HAVING COUNT(*) > 5)
  AND constructorId NOT IN (SELECT results.constructorId
                            FROM results
                                INNER JOIN races ON races.raceId = results.raceId
                            WHERE races.year > 2010)
```

22.
```
SELECT races.raceId, races.name, races.year, COUNT(DISTINCT results.statusId)
FROM races
    INNER JOIN results ON results.raceId = races.raceId
WHERE results.statusId BETWEEN 2 AND 7
GROUP BY races.raceId, races.name, races.year
HAVING COUNT(DISTINCT results.statusId) = (SELECT COUNT(*)
                                          FROM status
                                          WHERE statusId BETWEEN 2 AND 7)
```

23.
```
SELECT forename, surname, circuits.name, races.year, miliseconds
FROM drivers, pitstops, races, circuits
WHERE drivers.driverId=pitstops.driverId
  AND pitstops.raceId=races.raceId
```

```
  AND races.circuitId=circuits.circuitId
  AND drivers.nationality='Spanish'
  AND miliseconds = (SELECT MIN(miliseconds)
                     FROM pitstops JOIN drivers ON pitstops.driverId=drivers.
                         driverId
                     WHERE drivers.nationality='Spanish');
```

24.
```
SELECT DISTINCT name
FROM constructors
    INNER JOIN results ON results.constructorId = constructors.constructorId
    INNER JOIN drivers ON drivers.driverId = results.resultId
WHERE constructors.nationality LIKE 'Italian'
  AND drivers.nationality LIKE 'Italian'
```

25.
```
SELECT drivers.forename, drivers.forename, COUNT(*)
FROM drivers
    INNER JOIN results ON results.driverId = drivers.driverId
    INNER JOIN status ON status.statusId = results.statusId
WHERE status.status LIKE 'Accident'
GROUP BY drivers.driverId, drivers.forename, drivers.surname
HAVING COUNT(*) >= ALL (SELECT COUNT(*)
                        FROM results
                            INNER JOIN status ON status.statusId = results.statusId
                        WHERE status.status LIKE 'Accident'
                        GROUP BY results.driverId)
```

26.
```
SELECT drivers.forename, drivers.surname
FROM drivers
    INNER JOIN qualifying ON qualifying.driverId = drivers.driverId
    INNER JOIN races ON races.raceId = qualifying.raceId
WHERE position <= 10
  AND year = 2015
GROUO BY drivers.driverId, drivers.forename, drivers.surname
HAVING COUNT(*) = (SELECT COUNT(*) FROM races WHERE year = 2015);
```

27.
```
SELECT name
FROM constructors
WHERE constructorId NOT IN (SELECT constructorId
                            FROM results JOIN drivers ON results.driverId=drivers.
                                driverId
                            WHERE nationality='Italian')
AND nationality='Italian';
```

28.
```
SELECT constructors.*
FROM constructors
    INNER JOIN qualifying ON qualifying.constructorId = constructors.constructorId
    INNER JOIN races ON races.raceId = qualifying.raceId
WHERE position <= 10
    AND year = 2006
GROUO BY constructors.constructorId
HAVING COUNT(DISTINCT raceId) = (SELECT COUNT(*) FROM races WHERE year = 2006);
```

29.
```
SELECT forename, surname, suma
FROM(SELECT driverId, year, SUM(points) as suma
        FROM results JOIN races ON results.raceId=races.raceId
        WHERE year BETWEEN 1990 AND 2000
        GROUP BY driverId, year
        HAVING SUM(points)>0) AS puntuaciones
JOIN drivers ON puntuaciones.driverId=drivers.driverId
WHERE suma >= ALL (SELECT SUM(points)
                   FROM results JOIN races ON results.raceId=races.raceId
                   WHERE year BETWEEN 1990 AND 2000
                   GROUP BY driverId, year
                   HAVING SUM(points)>0)
```

30.
```sql
SELECT DISTINCT drivers.forename, drivers.surname, races.name, races.year
FROM drivers
    INNER JOIN results ON results.driverId = drivers.driverId
    INNER JOIN races ON races.raceId = results.raceId
WHERE results.positionOrder = 1
  AND driverId IN (SELECT driverId
                   FROM qualifying
                   WHERE qualifying.raceId = races.raceId
                     AND position > 10)
```

31.
```sql
SELECT drivers.forename, drivers.surname, COUNT(*)
FROM drivers
    INNER JOIN pitstops ON pitstops.driverId = drivers.driverId
    INNER JOIN races ON races.raceId = pitstops.raceId
WHERE races.year = 2013
GROUP BY pitstops.driverId, pitstops.raceId, drivers.forename, drivers.surname
HAVING COUNT(*) >= ALL (SELECT COUNT(*)
                        FROM pitstops
                        GROUP BY pitstops.driverId, pitstops.raceId)
```

32.
```sql
SELECT drivers.forename, drivers.surname
FROM drivers
    INNER JOIN results ON results.driverId = drivers.driverId
    INNER JOIN races ON races.raceId = results.raceId
WHERE year = 2017
  AND positionOrder <= 10
GROUP BY drivers.driverId, drivers.forename, drivers.surname
HAVING COUNT(*) = (SELECT COUNT(*) FROM races WHERE year = 2017)
```

Procedimientos almacenados:

1.
```sql
DELIMITER $$
CREATE PROCEDURE `getRacesInAYear`(IN `year` INTEGER)
BEGIN
  SELECT races.name, COUNT(DISTINCT results.constructorId) AS numConstructors
    FROM results
      INNER JOIN races ON races.raceId = results.raceId
    WHERE races.year = year
    GROUP BY races.raceId;
END$$
DELIMITER ;
```

2.
```sql
DELIMITER $$
CREATE PROCEDURE getsOnRaceMessages(IN cod VARCHAR(3), OUT msg VARCHAR(200))
BEGIN
    CASE cod
    WHEN 'E01' THEN SET msg = 'Error en la presion de las ruedas';
    WHEN 'E02' THEN SET msg = 'Pinchazo';
    WHEN 'E03' THEN SET msg = 'Temperatura alta en el motor';
    WHEN 'E04' THEN SET msg = 'Frenos sobre-calentados';
    WHEN 'E05' THEN SET msg = 'Error presion del aceite';
    ELSE SET msg = 'Error de comando';
    END CASE;
END$$
DELIMITER ;
```

3.
```sql
DELIMITER $$
CREATE PROCEDURE driversWinningAtHome (IN year_win INTEGER)
BEGIN
    SELECT DISTINCT drivers.forename, drivers.surname, circuits.name
    FROM drivers INNER JOIN results ON drivers.driverId = results.driverId INNER
        JOIN races ON results.raceId=races.raceId INNER JOIN constructors ON results
        .constructorId=constructors.constructorId INNER JOIN circuits ON circuits.
        circuitId=races.circuitId
    WHERE results.positionOrder=1 AND races.year=year_win AND drivers.nationality=
        constructors.nationality;
END$$
DELIMITER ;
```

4.
```
DELIMITER $$
CREATE PROCEDURE allPodiumPositions(IN anyo INTEGER)
BEGIN
SELECT forename, surname
FROM drivers D
WHERE driverId IN(SELECT driverId
                  FROM results JOIN races ON results.raceId=races.raceId
                  WHERE positionOrder=1
                    AND year=anyo)
  AND driverId IN(SELECT driverId
                  FROM results JOIN races ON results.raceId=races.raceId
                  WHERE positionOrder=2
                    AND year=anyo)
  AND driverId IN(SELECT driverId
                  FROM results JOIN races ON results.raceId=races.raceId
                  WHERE positionOrder=3
                    AND year=anyo);
END$$
DELIMITER ;
```

5.
```
DELIMITER $$
CREATE PROCEDURE getsConstructoresYPilotos (IN año YEAR)
BEGIN
    SELECT constructors.name, drivers.surname, SUM(results.points)
    FROM constructors JOIN results ON constructors.constructorId = results.
        constructorID
        JOIN races ON results.raceId = races.raceId
        JOIN drivers ON results.driverId = drivers.driverId
    WHERE races.year= año
    GROUP BY constructors.name, drivers.surname;
END$$
DELIMITER ;
```

6.
```
DELIMITER $$
CREATE PROCEDURE getNumberOfVictories (IN type VARCHAR(20))
BEGIN
    IF type = 'nationality' THEN
        SELECT drivers.nationality, COUNT(*) AS numVictorires
        FROM results
            INNER JOIN drivers ON drivers.driverId = results.driverId
        WHERE results.positionOrder = 1
        GROUP BY drivers.nationality
        ORDER BY numVictorires DESC;
    ELSE
        SELECT constructors.name, COUNT(*) AS numVictorires
        FROM results
            INNER JOIN constructors ON constructors.constructorId = results.
                constructorId
        WHERE results.positionOrder = 1
        GROUP BY constructors.name
        ORDER BY numVictorires DESC;
    END IF;
END$$
DELIMITER ;
```

7.
```
ALTER TABLE drivers ADD COLUMN añosEnActivo INTEGER NULL;

DELIMITER $$
CREATE PROCEDURE actualizarAñosEnActivo ()
BEGIN
    DECLARE done INTEGER DEFAULT FALSE;
    DECLARE id INTEGER;

    DECLARE cur CURSOR FOR SELECT driverId FROM drivers;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO id;
        IF done THEN
            LEAVE read_loop;
        END IF;
```

```
        UPDATE drivers
        SET drivers.añosEnActivo = añosEnActivo(id)
        WHERE driverId = id;
    END LOOP;
    CLOSE cur;
END$$
DELIMITER ;
```

8.
```
DELIMITER $$
CREATE PROCEDURE getDriversByNationality (IN nat VARCHAR(250), OUT drvs TEXT)
BEGIN
    DECLARE done INTEGER DEFAULT FALSE;
    DECLARE primer INT DEFAULT TRUE;
    DECLARE f, s VARCHAR(100);

    DECLARE cur CURSOR FOR SELECT forname, surname
                           FROM drivers
                           WHERE nationality = nat;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO id;
        IF done THEN
            LEAVE read_loop;
        END IF;

        IF primerDriver THEN
            SET drvs = CONCAT(f, ' ', s);
            SET primerDriver = FALSE;
        ELSE
            SET drvs = CONCAT(', ', f, ' ', s);
        END IF;
    END LOOP;
    CLOSE cur;
END$$
DELIMITER ;
```

Funciones almacenadas

1.
```
DELIMITER $$
CREATE FUNCTION puntosCampeon (year INTEGER)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE points DECIMAL(10,2);

    SELECT MAX(T.totalPoints) INTO points
    FROM (SELECT SUM(results.points) AS totalPoints
            FROM results
                INNER JOIN races ON races.raceId = results.raceId
            WHERE races.year = year
            GROUP BY results.driverId) AS T;

    RETURN (points);
END$$
DELIMITER ;
```

2.
```
DELIMITER $$
CREATE FUNCTION mediaPuntosConstructor (constructor VARCHAR(200))
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE points DECIMAL(10,2);

    SELECT AVG(T.totalPoints) INTO points
    FROM (SELECT SUM(results.points) AS totalPoints
            FROM results
```

```
                INNER JOIN races ON races.raceId = results.raceId
                INNER JOIN constructors ON constructors.constructorId = results.
                    constructorId
            WHERE constructors.name = constructor
            GROUP BY races.year) AS T;

    RETURN (points);
END$$
DELIMITER ;
```

3.
```
DELIMITER $$
CREATE FUNCTION añosEnActivo (id INTEGER)
RETURNS INTEGER
DETERMINISTIC
BEGIN
    DECLARE años INTEGER;

    SELECT COUNT(DISTINCT races.year) INTO años
    FROM results
        INNER JOIN races ON races.raceId = results.raceId
    WHERE results.driverId = id;

    RETURN (años);
END$$
DELIMITER ;
```

4.
```
DELIMITER $$
CREATE FUNCTION diffPoints (driver1 INTEGER, driver2 INTEGER)
RETURNS DOUBLE
DETERMINISTIC
BEGIN
    DECLARE points1 DOUBLE;
    DECLARE points2 DOUBLE;

    SELECT SUM(points) INTO points1
    FROM results
    WHERE driverId = driver1;

    SELECT SUM(points) INTO points2
    FROM results
    WHERE driverId = driver2;

    RETURN (points1 - points2);
END$$
DELIMITER ;
```

Triggers

1.
```
-- Se asumen que un piloto NUNCA va a participar 2 veces en la misma carrera

DELIMITER $$
CREATE TRIGGER noMasDeDosPilotos
BEFORE INSERT ON results
FOR EACH ROW
BEGIN
    DECLARE numDrivers INTEGER;

    SELECT COUNT(*) INTO numDrivers
    FROM results
    WHERE constructorId = NEW.constructorId
      AND raceId = NEW.raceId;

    IF numResults >= 2 THEN
        SIGNAL SQLSTATE '03000'
        SET MESSAGE_TEXT = 'Error: no pueden participar mas de dos pilotos por
            equipo';
    END IF;
END$$
DELIMITER ;
```

```
DELIMITER $$
CREATE TRIGGER noMasDeDosPilotos
BEFORE UPDTE ON results
FOR EACH ROW
BEGIN
    IF NEW.cronstructorId <> OLD.constructorId OR NEW.raceId <> OLD.raceId THEN

        DECLARE numDrivers INTEGER;

        SELECT COUNT(*) INTO numDrivers
        FROM results
        WHERE constructorId = NEW.constructorId
        AND raceId = NEW.raceId;

        IF numResults >= 2 THEN
            SIGNAL SQLSTATE '03000'
            SET MESSAGE_TEXT = 'Error: no pueden participar mas de dos pilotos por
                equipo';
        END IF;
    END IF;
END$$
DELIMITER ;
```

2.
```
CREATE TABLE crashes (
    crashId INTEGER UNIQUE NOT NULL AUTO_INCREMENT,
    driverId INT NOT NULL,
    description VARCHAR(250) DEFAULT NULL,
    PRIMARY KEY (crashId),
    CONSTRAINT
        FOREIGN KEY (driverId)
        REFERENCES drivers (driverId)
)

 DELIMITER $$
 CREATE TRIGGER resgistrarAccidentes
 AFTER INSERT ON results
 FOR EACH ROW
 BEGIN
     IF NEW.statusId = 3 OR NEW.statusId = 4 THEN
         INSERT INTO crashes (driverId, description) VALUES (NEW.driverId, 'blah
             blah blah');
     END IF;
 END$$
 DELIMITER ;
```

3.
```
DELIMITER $$
CREATE TRIGGER no_more_than_one_teams_in_a_year
BEFORE INSERT ON results
FOR EACH ROW
BEGIN
    DECLARE y YEAR;
    DECLARE num_races INTEGER;
    DECLARE num_races_with_constructor INTEGER;

    SELECT year INTO y FROM races WHERE raceId = NEW.raceId;

    SELECT COUNT(*), COUNT(constructorId = NEW.constructorId) INTO num_races,
        num_races_with_constructor
    FROM results
        INNER JOIN races ON races.raceId = results.raceId
    WHERE races.year = y
        AND driverId = NEW.driverId;

    IF num_races > 0 AND num_races_with_constructor = 0 THEN
        SIGNAL SQLSTATE '03000'
        SET MESSAGE_TEXT = 'Error: no se puede competir con un equipo con el que no
            hayas competido en ese año';
    END IF;
END$$
DELIMITER ;
```

```
DELIMITER $$
CREATE TRIGGER no_more_than_two_teams_in_a_year
BEFORE UPDATE ON results
FOR EACH ROW
BEGIN
    DECLARE y YEAR;
    DELCARE num_races INTEGER;
    DELCARE num_races_with_constructor INTEGER;

    IF NEW.raceId <> OLD.raceId OR NEW.driverId <> OLD.driverId OR NEW.constructorId
         <> OLD.constructorId THEN
        SELECT year INTO y FROM races WHERE raceId = NEW.raceId;

        SELECT COUNT(*), COUNT(constructorId = NEW.constructorId) INTO num_races,
            num_races_with_constructor
        FROM results
            INNER JOIN races ON races.raceId = results.raceId
        WHERE races.year = y
        AND driverId = NEW.driverId;

        IF num_races > 0 AND num_races_with_constructor = 0 THEN
            SIGNAL SQLSTATE '03000'
            SET MESSAGE_TEXT = 'Error: no se puede competir con un equipo con el que
                 no hayas competido en ese año';
        END IF;
    END IF;
END$$
DELIMITER ;
```

4.
```
CREATE TABLE sponsors(
    sponsorId INTEGER UNIQUE NOT NULL,
    name VARCHAR(50) NOT NULL,
    type VARCHAR(20),
    amount INTEGER NOT NULL,
    raceId INTEGER,
    PRIMARY KEY(sponsorId),
    CONSTRAINT
    FOREIGN KEY(raceId)
    REFERENCES formula1.races(raceId)
);

DELIMITER //
CREATE TRIGGER check_spon1 BEFORE INSERT ON sponsors
FOR EACH ROW
BEGIN
    IF NEW.amount>5000000 THEN SET NEW.type='Oficial';
    ELSE SET NEW.type='Co-oficial';
END IF;
END//

DELIMITER //
CREATE TRIGGER check_spon2 BEFORE UPDATE ON sponsors
FOR EACH ROW
BEGIN
    IF NEW.amount>5000000 THEN SET NEW.type='Oficial';
    ELSE SET NEW.type='Co-oficial';
END IF;
END//
DELIMITER ;
```

Titanic Spaceship

Consultas SQL

1.
```
SELECT nombre, CONCAT(cubierta, '-', numero_cabina, '-', lado_cabina),
    planeta_destino, sistema_destino
FROM pasajeros
WHERE cubierta IN (SELECT letra FROM cubiertas WHERE clase = 3)
  AND cubierta IN ('A', 'B', 'C', 'D')
```

```
  AND criosueño = 0;
```

2.
```
SELECT entretenimientos.nombre, SUM(gastos.cantidad), 'VIP'
FROM entretenimientos
    INNER JOIN gastos ON gastos.entretenimiento = entretenimientos.id
    INNER JOIN pasajeros ON pasajeros.id = gastos.pasajero
WHERE pasajeros.vip = 1
GROUP BY entretenimientos.id, entretenimientos.nombre;
```

3.
```
SELECT planeta_natal, sistema_natal, COUNT(*)
FROM pasajeros
GROUP BY planeta_natal, sistema_natal
HAVING COUNT(*) < (SELECT AVG (num_nacimientos)
                   FROM (SELECT COUNT(*) AS num_nacimientos
                         FROM pasajeros
                         GROUP BY planeta_natal, sistema_natal) t);
```

4.
```
SELECT pasajeros.nombre, entretenimientos.nombre, SUM(gastos.cantidad)
FROM entretenimientos
    INNER JOIN gastos ON gastos.entretenimiento = entretenimientos.id
    INNER JOIN pasajeros ON pasajeros.id = gastos.pasajero
WHERE pasajeros.cubierta = 'A'
  AND pasajeros.id IN (SELECT pasajero
                       FROM gastos
                       GROUP BY pasajero
                       HAVING COUNT(*) = (SELECT COUNT(*) FROM entretenimientos))
GROUP BY pasajeros.id, entretenimientos.id,
         pasajeros.nombre, entretenimientos.nombre
ORDER BY pasajeros.nombre, SUM(gastos.cantidad);
```

5.
```
SELECT cubierta, numero_cabina, lado_cabina
FROM pasajeros
WHERE transportado = 1
  AND id IN (SELECT pasajero
             FROM gastos
             WHERE entretenimiento = (SELECT entretenimiento
                                      FROM gastos
                                      GROUP BY entretenimiento
                                      HAVING COUNT(*)
                                          >= ALL (SELECT COUNT(*)
                                                  FROM gastos
                                                  GROUP BY entretenimiento)));
```

6.
```
SELECT DISTINCT planeta_destino, sistema_destino
FROM pasajeros
WHERE vip = 0
  AND (planeta_destino, sistema_destino) IN (SELECT planeta_destino, sistema_destino
                                             FROM pasajeros
                                             WHERE edad >= ALL (SELECT edad FROM
                                                 pasajeros));
```

7.
```
SELECT DISTINCT cubierta, numero_cabina, lado_cabina
FROM pasajeros
    INNER JOIN gastos ON gastos.pasajero = pasajeros.id
WHERE vip = 0
GROUP BY cubierta, numero_cabina, lado_cabina
HAVING COUNT(*) > 3
   AND SUM(gastos.cantidad)
       > (SELECT AVG(gasto_cabina) * 2
          FROM (SELECT SUM(cantidad) AS gasto_cabina
                FROM pasajeros
                    INNER JOIN gastos ON gastos.pasajero = pasajeros.id
                GROUP BY cubierta, numero_cabina, lado_cabina) t);
```

8.
```
SELECT planeta_natal, sistema_natal, COUNT(*) AS num_pasajeros_4_ent, num_pasajeros
FROM pasajeros
  NATURAL JOIN (SELECT planeta_natal, sistema_natal, COUNT(*) AS num_pasajeros
                FROM pasajeros
```

```
                    GROUP BY planeta_natal, sistema_natal) t
WHERE id IN (SELECT pasajero
             FROM gastos
             GROUP BY pasajero
             HAVING COUNT(DISTINCT entretenimiento) >= 4)
GROUP BY planeta_natal, sistema_natal
HAVING COUNT(*) >= (num_pasajeros / 4);
```

Vistas

1.
```
CREATE VIEW entretenimiento_pasajeros
AS SELECT id, pasajeros.nombre, edad, transportado, planeta_destino, sistema_destino
    , cubierta, numero_cabina, lado_cabina, cantidad, t.nombre AS entretenimiento
   FROM pasajeros
       INNER JOIN (SELECT pasajero, cantidad, nombre
                   FROM gastos g1
                       INNER JOIN entretenimientos ON entretenimientos.id = g1.
                           entretenimiento
                   WHERE cantidad = (SELECT MAX(cantidad)
                                     FROM gastos g2
                                     WHERE g2.pasajero = g1.pasajero)) t
         ON pasajeros.id = t.pasajero;
```

Procedimientos almacenados

1.
```
ALTER TABLE pasajeros ADD COLUMN tutor VARCHAR(7) NULL;
ALTER TABLE pasajeros ADD CONSTRAINT FOREIGN KEY (tutor) REFERENCES pasajeros(id);

DELIMITER $$
DROP PROCEDURE IF EXISTS asignar_tutores;
CREATE PROCEDURE asignar_tutores()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE id_menor VARCHAR(7);
    DECLARE id_tutor VARCHAR(7);

    DECLARE cur CURSOR FOR
        SELECT id FROM pasajeros WHERE edad < 18 AND criosueño = 0;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO id_menor;
        IF done THEN
            LEAVE read_loop;
        END IF;

        SELECT id INTO id_tutor
        FROM pasajeros
        WHERE edad >= 18
          AND (numero_cabina, lado_cabina, cubierta) =
            (SELECT numero_cabina, lado_cabina, cubierta
             FROM pasajeros
             WHERE id = id_menor)
        LIMIT 1;

        SET done = FALSE;

        IF id_tutor IS NOT NULL THEN
            UPDATE pasajeros SET tutor = id_tutor WHERE id = id_menor;
        ELSE
            UPDATE pasajeros SET criosueño = 1 WHERE id = id_menor;
        END IF;
    END LOOP;

    CLOSE cur;
END $$
DELIMITER ;
```

Funciones almacenadas

1.
```sql
DELIMITER $$
DROP FUNCTION IF EXISTS ocupacion;
CREATE FUNCTION ocupacion(letra_cubierta VARCHAR(1), planeta VARCHAR(50), sistema
    VARCHAR(50))
RETURNS FLOAT
DETERMINISTIC
BEGIN
   DECLARE pasajeros_cubierta INT;
   DECLARE total_pasajeros INT;

   SELECT COUNT(DISTINCT numero_cabina, lado_cabina) INTO pasajeros_cubierta
   FROM pasajeros
   WHERE cubierta = letra_cubierta
     AND planeta_destino = planeta
     AND sistema_destino = sistema;

   SELECT COUNT(*) INTO total_pasajeros
   FROM cabinas
   WHERE cubierta = letra_cubierta;

   RETURN (pasajeros_cubierta / total_pasajeros);
END$$
DELIMITER ;

SELECT DISTINCT cubierta, ocupacion(cubierta, '55 Cancri e', 'Copernico')
FROM pasajeros
WHERE planeta_destino = '55 Cancri e'
  AND sistema_destino = 'Copernico'
ORDER BY cubierta ASC;
```

Triggers

1.
```sql
DELIMITER $$
CREATE TRIGGER evitar_insertar_gasto_negativo BEFORE INSERT ON gastos
FOR EACH ROW
BEGIN
 IF (NEW.cantidad < 0) THEN
    SIGNAL SQLSTATE '02000'
    SET MESSAGE_TEXT = 'ERROR: Bienvenido ciber-delincuente, pero no puedes sacar
        dinero en un entretenimiento.';
 END IF;
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER evitar_actualizar_gasto_negativo BEFORE UPDATE ON gastos
FOR EACH ROW
BEGIN
 IF (NEW.cantidad < 0) THEN
    SIGNAL SQLSTATE '02000'
    SET MESSAGE_TEXT = 'ERROR: Bienvenido ciber-delincuente, pero no puedes sacar
        dinero en un entretenimiento.';
 END IF;
END$$
DELIMITER ;
```

2.
```sql
DELIMITER $$
CREATE TRIGGER promocionar_vip_insertar BEFORE INSERT ON gastos
FOR EACH ROW
BEGIN
    DECLARE total_gastado DOUBLE;

    SELECT SUM(cantidad) INTO total_gastado
    FROM gastos
    WHERE pasajero = NEW.pasajero;

    IF (total_gastado + NEW.cantidad) > 5000 THEN
```

```
        UPDATE pasajeros SET vip = 1 WHERE id = NEW.pasajero;
    END IF;
END$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER promocionar_vip_actualizar BEFORE UPDATE ON gastos
FOR EACH ROW
BEGIN
    DECLARE total_gastado DOUBLE;

    SELECT SUM(cantidad) INTO total_gastado
    FROM gastos
    WHERE pasajero = NEW.pasajero;

    IF (total_gastado - OLD.cantidad + NEW.cantidad) > 5000 THEN
        UPDATE pasajeros SET vip = 1 WHERE id = NEW.pasajero;
    END IF;
END$$
DELIMITER ;
```

Artículos científicos

Consultas SQL

1. ```
   SELECT distinct author_name
   FROM author a
       INNER JOIN author_affiliation aaf ON a.author_id = aaf.author_id
       INNER JOIN affiliation af ON aaf.affiliation_id = af.affiliation_id
       INNER JOIN author_article aar ON a.author_id = aar.author_id
       INNER JOIN article ar ON ar.DOI = aar.DOI
   WHERE af.affiliation_name LIKE '%Universidad%Politecnica%Madrid%'
     AND (YEAR(publication_date) = 2020 OR YEAR(publication_date) = 2021)
   ORDER BY author_name;
   ```

2. ```
   SELECT distinct author_name
   FROM author a
       INNER JOIN author_affiliation aaf ON a.author_id = aaf.author_id
       INNER JOIN affiliation af ON aaf.affiliation_id = af.affiliation_id
       INNER JOIN author_article aar ON a.author_id = aar.author_id
       INNER JOIN article ar ON ar.DOI = aar.DOI
   WHERE af.affiliation_name LIKE '%Universidad%Politecnica%Madrid%'
     AND YEAR(publication_date) = 2020
     AND a.author_id IN (SELECT author_id
                         FROM author_article aar
                             INNER JOIN article ar ON ar.DOI = aar.DOI
                         WHERE YEAR(ar.publication_date) = 2021)
   ORDER BY author_name;
   ```

3. ```
   SELECT distinct author_name, af.affiliation_name
   FROM author a
       INNER JOIN author_affiliation aaf ON a.author_id = aaf.author_id
       INNER JOIN affiliation af ON aaf.affiliation_id = af.affiliation_id
   WHERE a.author_id NOT IN (SELECT author_id
                             FROM author_article aar
                                 INNER JOIN article ar ON aar.DOI = ar.DOI
                             WHERE YEAR(publication_date) = 2020)
     AND a.author_id NOT IN (SELECT author_id
                             FROM author_article aar
                                 INNER JOIN article ar ON aar.DOI = ar.DOI
                             WHERE YEAR(ar.publication_date) = 2021)
     AND country_name = 'Spain'
   ORDER BY af.affiliation_name DESC, author_name;
   ```

4. ```
   SELECT journal_name, issn, SUM(num_citations) total_citas
   FROM journal j INNER JOIN article a ON j.journal_id = a.journal_id
   WHERE JIF_Quartile = 'Q1'
   ```

```
GROUP BY journal_name, issn
ORDER BY total_citas DESC;
```

5.
```
SELECT journal_name, SUM(num_citations) total_citas
FROM journal j INNER JOIN article a ON j.journal_id = a.journal_id
WHERE JIF_Quartile = 'Q1'
GROUP BY journal_name
HAVING SUM(num_citations) >= ALL (SELECT SUM(num_citations)
                                  FROM journal j INNER JOIN article a
                                      ON j.journal_id = a.journal_id
                                  WHERE JIF_Quartile = 'Q1'
                                  GROUP BY journal_name);
```

6.
```
SELECT affiliation_name
FROM affiliation a
    INNER JOIN author_affiliation aaf ON a.affiliation_id = aaf.affiliation_id
WHERE aaf.author_id IN (SELECT au.author_id
                        FROM author au
                            INNER JOIN author_article aua
                                ON au.author_id = aua.author_id
                            INNER JOIN article ar
                                ON aua.DOI = ar.DOI
                        WHERE YEAR(ar.publication_date) =
                            (SELECT MAX(YEAR(publication_date))
                             FROM article)
                        GROUP BY au.author_id
                        HAVING COUNT(DISTINCT ar.DOI) >=5)
GROUP BY affiliation_name
HAVING COUNT(DISTINCT aaf.author_id) >= 10;
```

7.
```
SELECT journal_name
FROM journal j
    INNER JOIN article a ON j.journal_id = a.journal_id
WHERE JIF > (SELECT AVG(JIF)
             FROM journal)
  AND YEAR(a.publication_date) = (SELECT MIN(YEAR(publication_date))
                                  FROM article)
GROUP BY journal_name
HAVING COUNT(DISTINCT a.DOI) >= 300;
```

8.
```
SELECT journal_name, total_citas
FROM journal j
    INNER JOIN article a ON j.journal_id = a.journal_id
    INNER JOIN (SELECT journal_id, SUM(num_citations) as total_citas
                FROM article
                GROUP BY journal_id) as journal_citations
                    ON j.journal_id = journal_citations.journal_id
GROUP BY j.journal_id, journal_name
HAVING total_citas > (SELECT AVG(total_citas)
                      FROM (SELECT SUM(num_citations) as total_citas
                            FROM article
                            GROUP BY journal_id) as T);
```

9.
```
SELECT author_name
FROM author a
    INNER JOIN author_article aa ON a.author_id = aa.author_id
    INNER JOIN article ar ON ar.DOI = aa.DOI
GROUP BY a.author_id, a.author_name
HAVING COUNT(DISTINCT YEAR(publication_date)) =
    (SELECT COUNT(DISTINCT YEAR(publication_date))
     FROM article);
```

Procedimientos almacenados

1.
```
CREATE PROCEDURE maxAuthors (IN year INT, OUT journal VARCHAR(300), OUT numauthors
    INT)
BEGIN
  DECLARE done bool DEFAULT FALSE;
  DECLARE j_id BIGINT;
  DECLARE j_name VARCHAR(300);
  DECLARE maxjournal VARCHAR(300);
  DECLARE currentAuthors, maxAuthors INT;
  DECLARE journal_cursor CURSOR FOR SELECT journal_id, journal_name FROM journal;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  OPEN journal_cursor;
  SET maxAuthors = 0;
  process_loop: LOOP
    FETCH journal_cursor into j_id, j_name;
    IF done THEN
      LEAVE process_loop;
    END IF;
    set currentAuthors = 0;
    set journal = '';

    SELECT DISTINCT COUNT(DISTINCT author_id) INTO currentAuthors
    FROM article a
        INNER JOIN author_article aa ON a.DOI = aa.DOI
    WHERE journal_id = j_id
      AND YEAR(publication_date) = year
    GROUP BY a.DOI, a.title
    HAVING COUNT(DISTINCT author_id) >= ALL (SELECT COUNT(DISTINCT author_id)
                                            FROM article a
                                                INNER JOIN author_article aa ON a.
                                                    DOI = aa.DOI
                                            WHERE journal_id = j_id
                                              AND YEAR(publication_date) = year
                                            GROUP BY a.DOI, a.title);

    IF currentAuthors > maxAuthors THEN
      SET maxAuthors = currentAuthors;
      SET maxjournal = j_name;
    ELSEIF currentAuthors = maxAuthors THEN
      SET maxjournal = CONCAT(maxjournal, '; ', j_name);
    END IF;
  END LOOP;

  CLOSE journal_cursor;
  SET numauthors = maxAuthors;
  SET journal = maxjournal;
END$$
DELIMITER ;
```

Funciones almacenadas

1.
```
DELIMITER $$
CREATE FUNCTION avgArticles (journal BIGINT)
RETURNS FLOAT
DETERMINISTIC
BEGIN
   DECLARE numArticles INT;
   DECLARE numanios INT;

   SELECT COUNT(DISTINCT DOI) INTO numArticles
   FROM article
   where journal_id = journal;

   SELECT COUNT(DISTINCT YEAR(publication_date)) INTO numanios
   FROM article;

   RETURN (numArticles/numanios);
END$$
DELIMITER ;
```

Triggers

1.
```sql
CREATE TABLE reviews (
    author_id BIGINT NOT NULL,
    DOI VARCHAR(200) NOT NULL,
    sending_date DATE,
    review_done BOOL,
    review_result SMALLINT,
    PRIMARY KEY (author_id, DOI),
    FOREIGN KEY (author_id) REFERENCES author (author_id),
    FOREIGN KEY (DOI) REFERENCES article (DOI)
);

DELIMITER $$
CREATE TRIGGER avoidReview BEFORE INSERT ON reviews
FOR EACH row
BEGIN
 IF (NEW.author_id IN (SELECT author_article.author_id FROM author_article WHERE DOI
      = NEW.DOI)) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'ERROR: No puede revisar un autor del articulo';
 END IF;
END$$
DELIMITER ;
```