TeraCode Hiring v2.2

Ejercicio 1

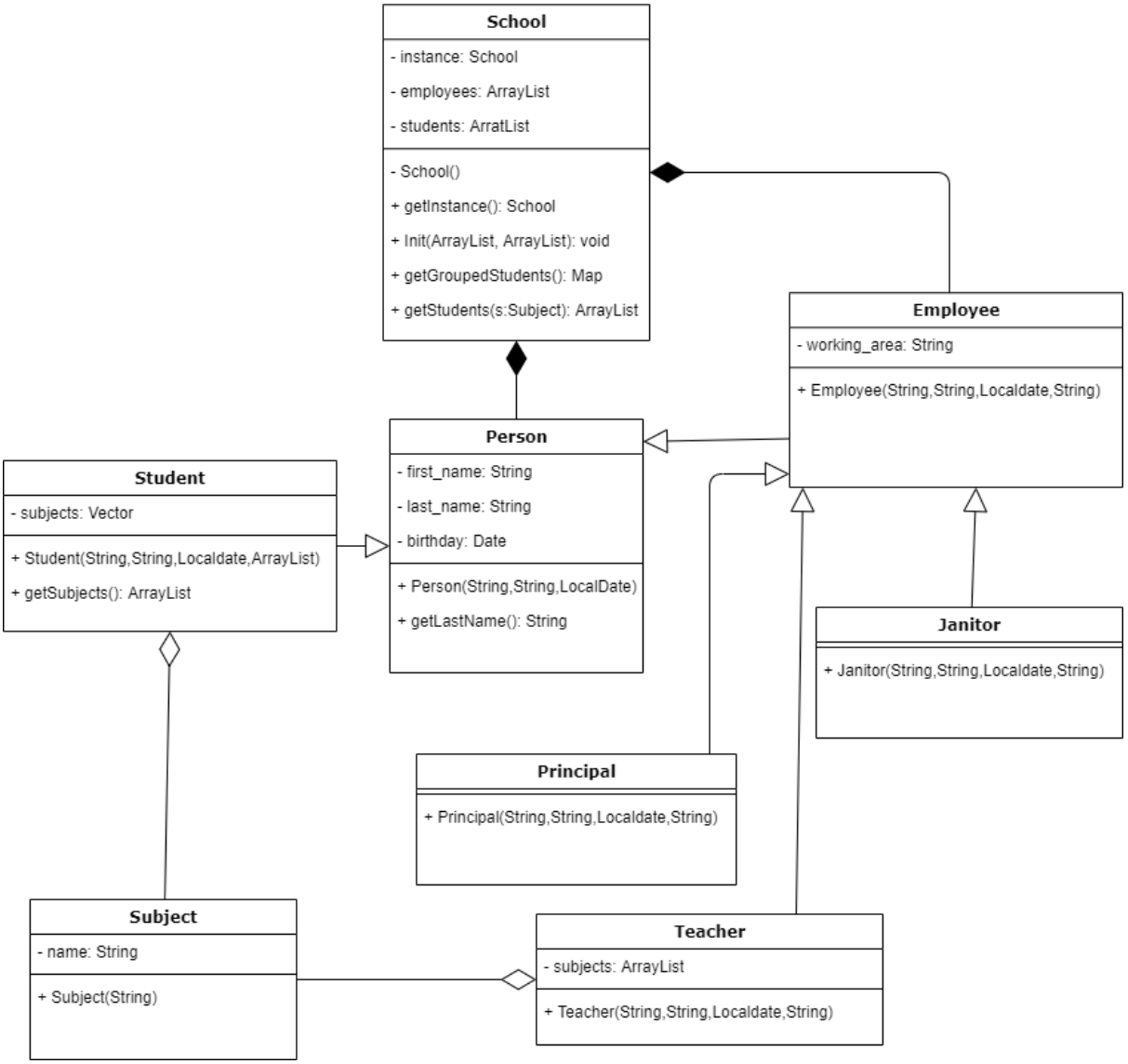
Acotaciones:

- Supongo que el director (principal) es un empleado ya que éste puede cambiar con los años o haber un suplente. Los directores son empleados en las escuelas públicas.
- Además de las entidades sugeridas, agregué Teacher para representar a los maestros y School como entidad raíz en el diagrama de clases.
- Asumo que todos los empleados (principal, teachers, janitors) tienen asignados un área de trabajo.
- Supongo todos los setters y getters correspondientes.
- Se asume que se va a utilizar con 1 sola escuela, por lo que utilizo el patrón de diseño Singleton para la clase raíz (con los métodos getInstance e Init) y la entidad School no será creada en la base de datos.

Inciso A

Acotaciones:

- Se solicita un diagrama del modelo de dominio y entidades, pero también habla de clases abstractas e interfaces. Supongo que es un diagrama de clases.



Inciso B

Acotaciones:

- Las agrupaciones se realizan sin distinguir mayúsculas ni minúsculas. P.ej. los apellidos “Pérez” y “pilar” estarán en el mismo grupo.

```
/**
 * Returns all students grouped by last name
 *
 * @return map
 */
public Map getGroupedStudents() {
    Map<Character, ArrayList<Student>> result = new HashMap<>();
    Character first_letter;
    for (Student it : this.students) {
        first_letter = Character.toUpperCase(it.getLastName().charAt(0));
        ArrayList<Student> grouped_students;
        if (!result.containsKey(first_letter)) {
            grouped_students = new ArrayList<>();
            result.put(first_letter, grouped_students);
        } else
            grouped_students = result.get(first_letter);
        grouped_students.add(it);
    }  // for

    return result;
}
```

Inciso C

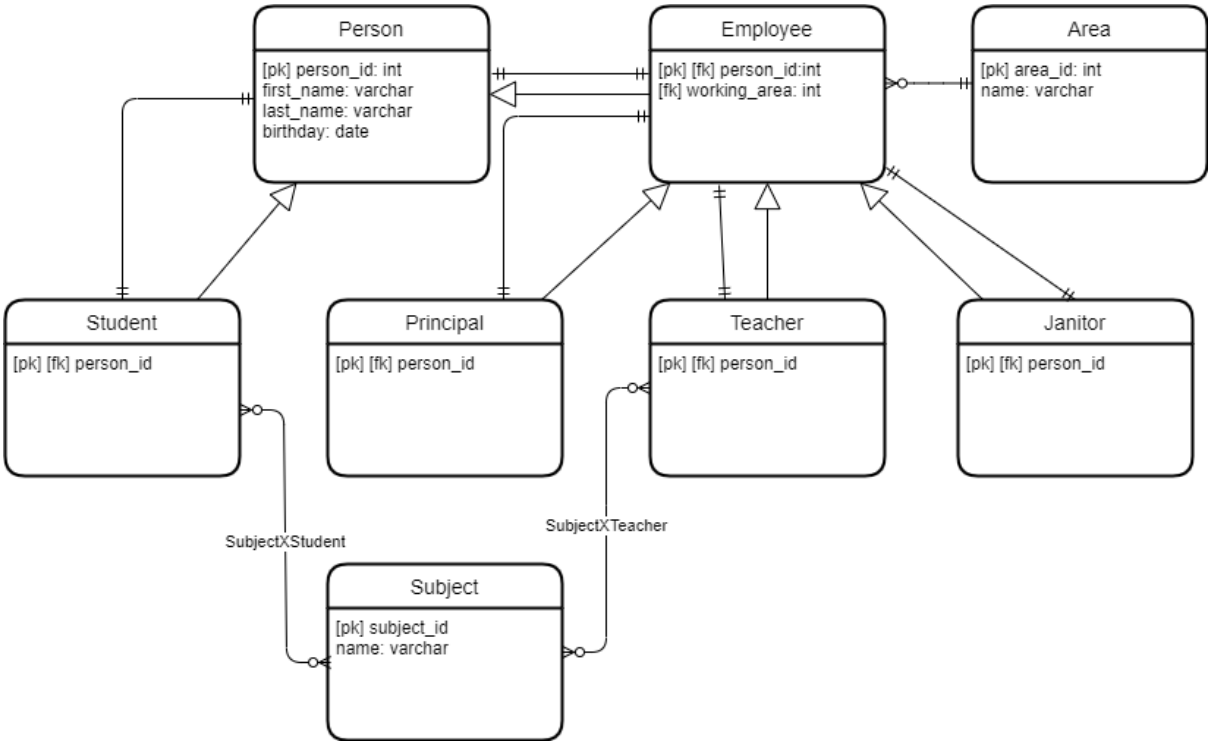
El modelo implementado no permite la repetición de alumnos en las materias, es decir que no se necesita ningún control para que se cumpla esta regla si el dataset de alumnos es correcto (no hay repetidos).
Se supone que en el dataset no se deben encontrar alumnos repetidos, esta verificación no debe ser responsabilidad de este método sino de la base de datos; o si no se cuenta con una base de datos, debe controlarlo el método que crea al alumno.

De todos modos, agrego el chequeo adicional por si el dataset de alumnos no se encuentra correctamente cargado.

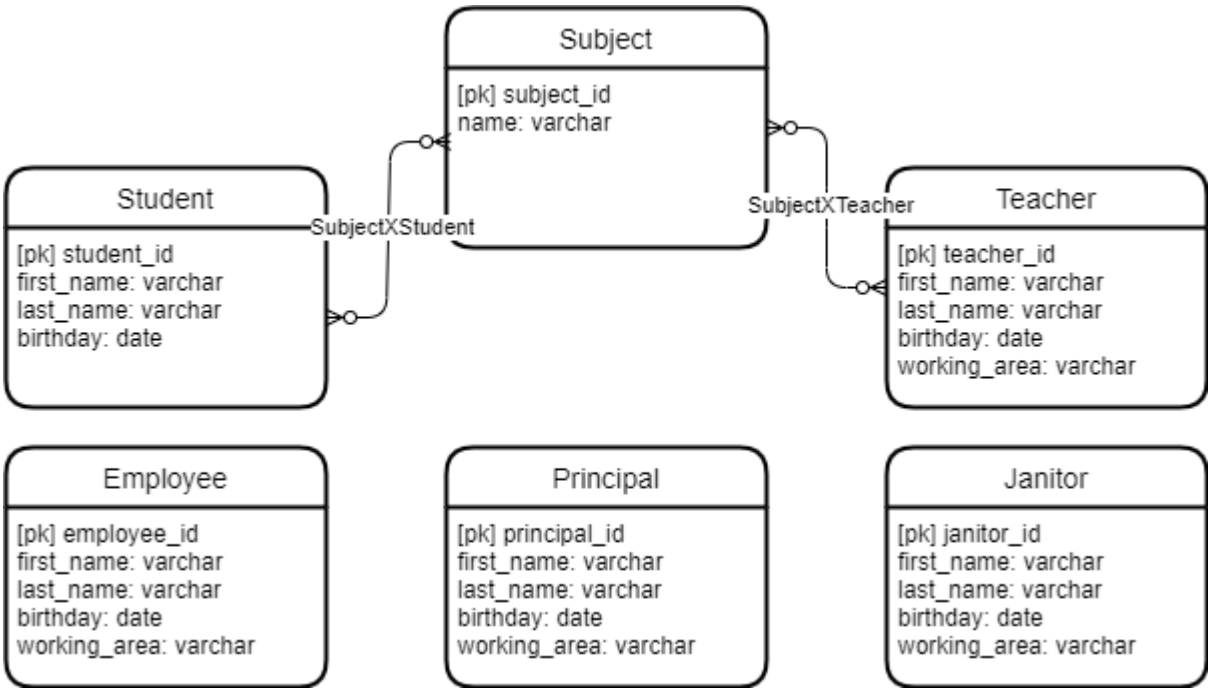
```
/**
 * Returns all students associated with a subject
 *
 * @param  subject a subject
 * @return array
 */
public ArrayList getStudents(Subject subject) {
    ArrayList<Student> result = new ArrayList<>();
    for (Student it : this.students)
        if (it.getSubjects().contains(subject) && !result.contains(it))
            result.add(it);

    return result;
}
```

Inciso D

ERD 1



ERD 2



En el modelo conceptual **ERD 1** se utilizó la técnica de generalización parcial y exclusiva, donde un registro de la entidad padre puede o no tener su correspondiente registro en la

entidad hija (parcial ya que pueden haber más tipos de empleados, como administrativos) y una persona no puede pertenecer a más de una subentidad (exclusiva).
Además, las áreas se definieron como una entidad en sí misma;, esto permitirá identificarlas y recuperar datos sin tener el problema de la duplicación de área o la falta de datos. P.ej. si se carga mal el atributo workingArea debido a que es texto libre, pueden faltar registros a la hora de recuperar datos asociados de un área particular.
Las ventajas de este modelo es la escalabilidad, el mantenimiento y la menor cantidad de atributos creados.
Las desventajas que pueden haber en este modelo están relacionadas con la pérdida de performance al momento de realizar los ensambles y chequear las restricciones (asserts); p.ej. para obtener el registro de un maestro, hay que realizar 3 ensambles.

En el **ERD 2** se aplanaron las tablas, quitando la técnica de generalización; de esta manera en las entidades especializadas se suman los atributos de las entidades generales. También se quitó la entidad que representa las áreas para que coincida con el ejemplo de consulta del **inciso E**.
Una de las ventajas de este modelo es la mayor eficiencia en la recuperación de la información debido a que no necesita realizar una gran cantidad de ensambles para recuperar registros de las entidades especializadas.
Las desventajas tienen que ver con la escalabilidad, mantenimiento y mayor cantidad de atributos en las entidades. Además, al desnormalizar el modelo quitando la entidad de áreas, tenemos el problema de la posibilidad de falta de unicidad en relación a esta información.

Cabe aclarar que en el modelo físico de ambos casos, las relaciones N a N entre Student, Teacher y Subject se transformarán en relaciones 1 a N con el agregado de 2 nuevas tablas (p.ej. SubjectXStudent y SubjectXTeacher) debido a que la gran mayoría de los motores de bases de datos no implementan relaciones N a N.

Inciso E

Los atributos “nombre” y “apellido” deberían pertenecer únicamente a la entidad Person, por lo tanto se puede quitar el ensamble con la entidad Employee, quedando la consulta de la siguiente manera:

```
SELECT * FROM janitor j
    INNER JOIN person p ON p.id = j.id
WHERE j.workingArea = 'Hallway';
```

Inciso F

La solución más sencilla sería aumentar el tamaño de la caché del motor de base de datos para que baje el nivel de invalidación y mantenga el ResultSet de la consulta, pero puede que el problema no sea tan sencillo.
A continuación se tratan varias alternativas de mejora.

Consulta y aislamiento

Uno de los problemas puede estar en la optimización de la consulta, teniendo la posibilidad de acelerar la misma con el uso de índices. Los índices pueden penalizar el rendimiento si las tablas involucradas se actualizan con frecuencia, pero si prima la lectura por sobre la escritura, el uso de índices puede mejorar considerablemente el tiempo de respuesta de las consultas.

El uso de aislamientos muy restrictivos también puede llevar a consultas lentas, sobre todo cuando tiene que esperar a liberar un registro bloqueado. Un nivel de aislamiento Read Commited es un valor razonable en la mayoría de las ocasiones.

En ambos casos, es recomendable desactivar la memoria caché del motor para realizar las pruebas de rendimiento.

Caché

La mayoría de motores de bases de datos administran su propia caché en memoria física para mejorar la latencia de las consultas. Si vemos que el tamaño de la caché es bajo, podemos configurar el motor aumentando este valor.

Si el servidor cuenta con poca memoria física, es recomendable agregar memoria y ajustar los parámetros del motor para que destine la mayor cantidad posible a la caché.

Servidor

Si notamos que el problema viene dado por la alta carga de I/O en el servidor, tenemos la posibilidad de mejorar la performance de acceso a los medios físicos mediante un arreglo de discos RAID. Éste puede ser por hardware (como un HBA SAS) o software (como MD-Raid en Linux).
La utilización de un RAID 0 con 6 discos aumentará casi 6 veces la velocidad de lectura ya que el driver permite la lectura de los sectores de cada disco en paralelo.

Si los discos rígidos son mecánicos, también podemos mitigar este problema con soluciones vía software que permiten utilizar un disco de estado sólido como caché. P.ej. en Linux se puede utilizar LVCache, EnhanceIO, etc..
Esto aumenta enormemente la cantidad de I/O que soportará el servidor.

Infraestructura

SCSI y Fiber-channel

Volviendo al caso de la implementación de RAID en el servidor, si quisiéramos utilizar un arreglo de discos aún mayor para mejorar aún más la latencia, podemos crear una red dedicada de almacenamiento de fibra óptica y utilizar el protocolo SCSI para crear arreglos casi ilimitados de discos. P.ej. un arreglo de 64 discos tendrá una mejora en la lectura de casi 64 veces.

Middleware

Así como los SGBD administran su propia caché, también podemos agregar a la arquitectura una capa intermedia de gestión de caché con soluciones como Memcached o Redis.
Para ello necesitamos serializar los ResultSets y guardarlos en la caché intermedia ya que esta clase de caché utiliza el par clave-valor.
El problema con este tipo de arquitectura es que la invalidación de los datos en caché debemos manejarla desde la aplicación, mientras que la caché del SGBD la gestiona el propio motor de base de datos, siendo completamente transparente.

Clustering

Si el problema de la demora se encuentra en la carga del servidor, se puede plantear el escalamiento horizontal en un cluster. La ventaja de escalar horizontalmente es que la carga se reparte entre todos los nodos que componen el cluster.

Existen 2 modelos para realizar un cluster de base de datos: master-master o master-slave.
En un modelo master-master, una transacción se completa cuando el datos se actualizan en todos los nodos master; mientras que en el modelo master-slave los nodos esclavos se utilizan para lectura.
Para nuestro caso particular, conviene utilizar el modelo master-slave debido a que éste permite lecturas asíncronas, aumentando considerablemente la velocidad en la recuperación de datos.
La ventaja de un cluster es que se pueden agregar tantos nodos como sean necesarios hasta que la carga se estabilice.

Último recurso

Si luego de haber probado todo lo anterior aún la consulta sigue demorando demasiado, podemos desnormalizar las tablas involucradas en la base de datos para mejorar la latencia.
Si bien no es recomendable utilizar esta técnica, hace algunos años atrás cuando el rendimiento de los motores de base de datos y el hardware no estaban tan optimizados, se realizaba esto en casos particulares cuando el costo / beneficio lo ameritaba.

Inciso G

Acotaciones:

- Supongo 19 y 21 inclusive, sino serían únicamente los estudiantes de 20 años de edad.

Si bien se podrían utilizar funciones propias del motor para calcular la edad a partir de la fecha de nacimiento en la propia consulta, lo ideal es crear una vista cuando son necesarias columnas derivadas.
En este caso creé una vista con la nueva columna “age” calculada desde la fecha de nacimiento.

```
# Código MySQL
CREATE VIEW StudentView AS
SELECT
    S.person_id,
    first_name,
    last_name,
    TIMESTAMPDIFF(YEAR, birthdate, curdate()) AS age
FROM Student S
    LEFT JOIN Person P ON P.person_id = S.person_id;

SELECT person_id, age
FROM StudentView
WHERE
    age BETWEEN 19 AND 21;
```

Inciso H

Se puede llevar la lógica del negocio a la base de datos utilizando Stored Procedures y llamando las funciones desde la aplicación, pero no es una práctica recomendable.
En una correcta división de responsabilidades, la lógica del negocio no debería estar en la base de datos, ya que ésta debería tener únicamente la lógica para manipular la persistencia y el acceso a los datos.

Si bien hay varias maneras de tratar este problema, uno de los diseños arquitectónicos que funciona bien y lleva varios años, es utilizar el patrón multinivel MVC junto a un ORM.
El patrón MVC divide las responsabilidades residiendo la lógica del negocio en el Modelo, y el ORM crea una capa adicional (Data Access Layer o DAL) que abstrae la persistencia y permite el tratamiento de los datos como objetos dentro del Modelo (MVC) de la aplicación, separando aún más las responsabilidades de la lógica del negocio con la lógica de acceso a los datos.

Como ejemplo de persistencia de un estudiante, suponemos que el usuario quiere crear un nuevo registro desde la aplicación. El proceso sería el siguiente:

1. El usuario llena los datos del estudiante en un formulario y hace clic en el botón Agregar.
2. El controlador captura el evento, realiza una verificación básica preliminar y llama al modelo para que cree y persista al nuevo estudiante, pasándole los datos por parámetro.
3. El modelo verifica los datos de acuerdo a las reglas de negocio y crea una nueva instancia del estudiante.
4. El ORM mapea los atributos de la nueva instancia con la entidad correspondiente en la base de datos y crea el registro.
5. El ORM también se encarga de la lógica funcional de los objetos relacionados con las entidades, generando los getters y setters correspondientes.

Los beneficios de este tipo de arquitectura es la mejora en el mantenimiento debido a la división de responsabilidades, la escalabilidad y la gestión de la persistencia orientada a

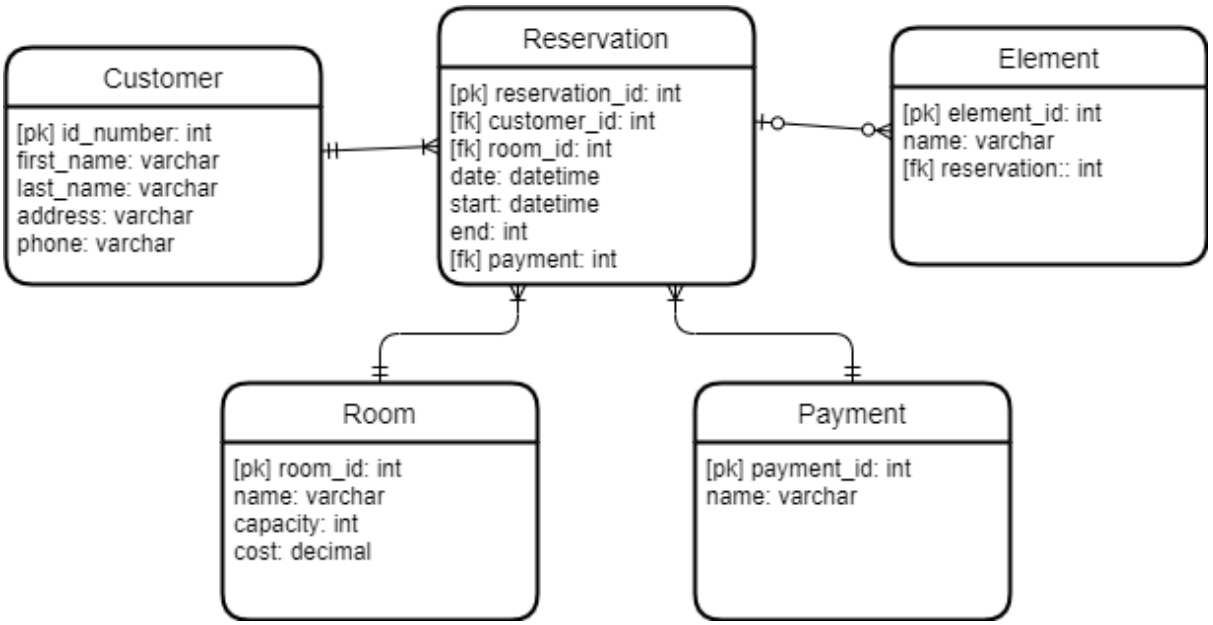
objetos, permitiendo al programador de la aplicación abstraerse de la base de datos (hace pocos años atrás el código SQL se embebía en la aplicación).
Otro claro beneficio es la abstracción del motor de base de datos, lo que permite migrar de un motor a otro (salto de versiones u otro motor completamente distinto) con un mínimo costo.

Pero también existen problemas, por ejemplo en el modelo MVC es la penalización de la performance debido a la mayor cantidad de indirecciones; también el tamaño que puede alcanzar el Modelo debido a que generalmente es la capa más compleja de la aplicación (hay variantes del diseño MVC donde el modelo se puede dividir en varias capas adicionales).
También existen problemas con los ORM debido a que no siempre se puede mapear un modelo relacional complejo a uno orientado a objetos, generando una complejidad muy grande y pérdida de rendimiento. Otro problema son las consultas no optimizadas ya que el código SQL generado por el ORM se realiza de manera automática y muchas veces es necesario intervenir manualmente para mejorar el rendimiento de las consultas.

Ejercicio 2

Inciso A

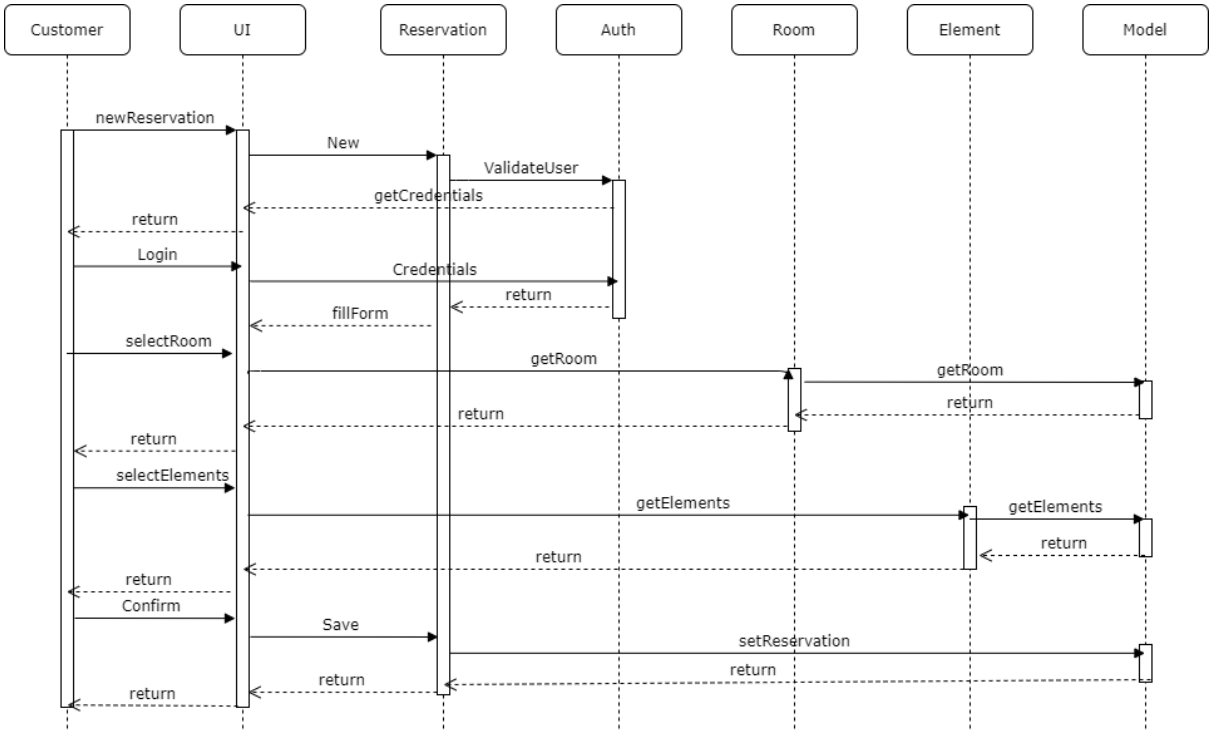
Customer: personas que alquilan la sala
Room: salas de conferencias
Payment; métodos de pago
Element: elementos que puede requerir el disertante (proyector, puntero láser, pizarra y fibrones, etc.)
Reservation: la reserva de una sala



Inciso B

Acotaciones:

- Estaba entre un diagrama de colaboración, de transición de estados, de secuencia o simplemente Storyboards mostrando la UI. Elegí un diagrama de secuencia para mostrar mejor la interacción entre los componentes.



Inciso C

Acotaciones:

- El comando SIGNAL SQLSTATE produce una excepción y revierte el proceso de inserción o actualización.
- La excepción puede ser capturada por la aplicación y mostrar el mensaje correspondiente al usuario.

```
CREATE TRIGGER before_reservation
    BEFORE INSERT OR UPDATE
    ON reservation FOR EACH ROW
BEGIN
    DECLARE duration INT;
    DECLARE overlap INT;

    -- Check the duration of the conference
    SET duration = TIMESTAMPDIFF(MINUTE, New.start, New.end);
    IF duration < 15 OR duration > 180 THEN
        SIGNAL SQLSTATE '45000' SET message_text = 'The duration must be less
than 15 or greater than 180 minutes';
    END IF;

    -- Check reservation overlap
    SET overlap = (
        SELECT Count(reservation_id)
        FROM reservation
        WHERE
             room_id = New.room_id AND
             start < New.end AND
             New.start < end
    );
    IF overlap > 0 THEN
        SIGNAL SQLSTATE '45000' SET message_text = 'There is a reservation in
that period';
    END IF;
END$$
```