

Especificación de requisitos de software

Proyecto: SQL Escape

Revisión [1.0.1]



Proyecto creado por:

Matías Otte <otte.matias@docente.ceibal.edu.uy>;

Sofía Rodríguez <sofi180403@gmail.com>;

Sol Méndez <bau21062014@gmail.com>;

Lucas Aguiar <holalucas17.arroyo@gmail.com>;

Agustín Muñoz <agustin.munoz.lagomarsino@gmail.com >.

Tutor:

Prof. Domingo Pérez <domingo.perez@1001problemas.com>



Contenido

1. Resumen Ejecutivo.....	3
1.1. Propósito del Proyecto.....	3
1.2. Alcance y Objetivos.....	3
1.3. Audiencia Objetivo.....	3
1.4. Arquitectura General.....	3
2. Análisis de Requisitos y Casos de Uso.....	3
2.1. Actores del Sistema.....	3
2.1.1. Jugador/Estudiante.....	3
2.1.2. Docente.....	4
2.1.3. Sistema de Evaluación.....	4
2.2. Diagrama de Casos de Uso.....	4
2.3. Especificación de Casos de Uso.....	6
2.3.1. UC01: Iniciar Partida.....	6
2.3.2. UC02: Resolver Desafío SQL.....	6
2.3.3. UC03: Validar Consulta SQL.....	7
2.3.4. UC04: Avanzar de Nivel.....	7
2.3.5. UC05: Guardar Progreso.....	7
3. Diseño de la Arquitectura Orientada a Objetos.....	14
3.1. Diagrama de Clases.....	14
3.2. Descripción de Clases Principales.....	0
3.2.1. SqlEscapeGame.....	0
3.2.2. Player y PlayerProgress.....	0
3.2.3. Level y Challenge.....	0
3.2.4. SqlEvaluator y SqlSyntaxValidator.....	0
3.2.5. GameDatabase y Clases DAO.....	0
3.3. Patrones de Diseño Aplicados.....	0
2 Descripción general.....	0
2.1 Perspectiva del producto.....	0
2.2 Funcionalidad del producto.....	0
2.3 Restricciones.....	0
2.4 Suposiciones y dependencias.....	0
2.5 Evolución previsible del sistema.....	0
3 Requisitos específicos.....	0
3.1 Requisitos comunes de los interfaces.....	0
3.1.1 Interfaces de usuario.....	0
3.1.4 Interfaces de comunicación.....	0
3.2 Requisitos funcionales.....	0
3.2.1 Requisito funcional 1.....	0



3.2.2 Requisito funcional 2.....	0
3.2.3 Requisito funcional 3.....	0
3.2.4 Requisito funcional n.....	0
3.3 Requisitos no funcionales.....	0
3.3.1 Requisitos de rendimiento.....	0
3.3.2 Seguridad.....	0
3.3.3 Fiabilidad.....	0
3.3.4 Disponibilidad.....	0
3.3.5 Mantenibilidad.....	0
3.3.6 Portabilidad.....	0
4 Análisis.....	0
4.1 Análisis de base de datos.....	0
4.1.1 Modelo Entidad-Relación (MER).....	0
a) Modelo Entidad-Relación del universo del juego (base de datos de los desafíos).....	0
b) Modelo Entidad-Relación del juego (gestión de usuarios, niveles y progreso).....	0
4.1.2 Modelo Relacional (MR).....	0
a) Proyección de nombres de marineros uruguayos.....	0
b) Listar barcos con más de 100 camarotes.....	0
c) Nombres de marineros que integran un barco.....	0
d) Información de barcos que han navegado hacia islas con puerto.....	0
e) Barcos con sensores en cubierta.....	0
4.1.3 DDL (Data Definition Language).....	0
4.1.4 DML (Data Manipulation Language).....	0
4.1.5 DCL (Data Control Language).....	0
4.1.6 TCL (Transaction Control Language).....	0
4.2 Análisis de programación.....	0
5 Diseño.....	0
6 Implementación.....	0
7 Apéndices.....	0



1. Resumen Ejecutivo

1.1. Propósito del Proyecto

SQL Escape es un videojuego educativo desarrollado en Java con el objetivo principal de facilitar el aprendizaje práctico del lenguaje SQL. El proyecto combina una narrativa inmersiva con desafíos técnicos para crear una experiencia de aprendizaje atractiva y efectiva para estudiantes y entusiastas de la tecnología.

1.2. Alcance y Objetivos

El juego consiste en una serie de al menos 10 niveles narrativamente conectados. En cada nivel, el jugador debe resolver un problema específico escribiendo y ejecutando consultas SQL. Los objetivos del proyecto son:

- **Enseñar SQL:** Proporcionar una plataforma interactiva para aprender y practicar consultas SELECT.
- **Fomentar la Resolución de Problemas:** Plantear desafíos que requieran análisis y lógica.
- **Crear una Experiencia Atractiva:** Utilizar una historia y elementos de juego para motivar al jugador.
- **Evaluar el Aprendizaje:** Implementar un sistema de evaluación automática que proporcione feedback instantáneo.

1.3. Audiencia Objetivo

- **Estudiantes:** Alumnos de cursos de bases de datos o programación.
- **Autodidactas:** Personas que deseen aprender SQL por su cuenta.
- **Docentes:** Profesores que busquen herramientas innovadoras para sus clases.

1.4. Arquitectura General

El sistema está diseñado con una **arquitectura orientada a objetos (OO)**, implementada en **Java**. La persistencia de datos se gestiona a través de una base de datos **MySQL**, administrada localmente con **XAMPP**. La gestión de dependencias y la construcción del proyecto se realizan con **Maven**.

2. Análisis de Requisitos y Casos de Uso

2.1. Actores del Sistema

2.1.1. Jugador/Estudiante

- **Descripción:** Es el usuario final que interactúa directamente con el juego. Su objetivo es progresar en la historia resolviendo los desafíos SQL.



2.1.2. Docente

- **Descripción:** Un actor secundario que puede utilizar el juego como una herramienta pedagógica. Podría tener acceso a los resultados y al progreso de los estudiantes para evaluar su desempeño.

2.1.3. Sistema de Evaluación

- **Descripción:** Un componente interno y automatizado del software. Es responsable de validar las consultas SQL, ejecutar las pruebas y determinar si las respuestas del jugador son correctas.

2.2. Diagrama de Casos de Uso

A continuación, se presenta el diagrama de casos de uso que ilustra las interacciones entre los actores y el sistema.

@startuml

!theme plain

title Diagrama de Casos de Uso - SQL Escape

left to right direction

actor "Jugador/Estudiante" as Jugador

actor "Docente" as Docente

actor "Sistema de Evaluación" as Sistema

rectangle "SQL Escape Game" {

 usecase "Iniciar Partida" as UC01

 usecase "Resolver Desafío SQL" as UC02

 usecase "Validar Consulta SQL" as UC03

 usecase "Avanzar de Nivel" as UC04

 usecase "Guardar Progreso" as UC05

 usecase "Consultar Historial" as UC06

 usecase "Reiniciar Partida" as UC07

 usecase "Visualizar Resultados" as UC08



```
usecase "Cargar Nivel" as UC09
usecase "Evaluar Respuesta" as UC10
usecase "Mostrar Narrativa" as UC11
usecase "Registrar Intento" as UC12
}
```

' Relaciones Jugador

Jugador --> UC01

Jugador --> UC06

Jugador --> UC07

' Relaciones Docente

Docente --> UC06

' Relaciones Sistema

Sistema --> UC03

' Inclusiones y extensiones

UC01 ..> UC09 : <<include>>

UC09 ..> UC02 : <<include>>

UC02 ..> UC03 : <<include>>

UC02 ..> UC08 : <<include>>

UC03 ..> UC10 : <<include>>

UC10 ..> UC04 : <<extend>>

UC04 ..> UC05 : <<include>>

UC04 ..> UC09 : <<include>>

UC02 ..> UC11 : <<include>>

UC03 ..> UC12 : <<include>>

@enduml

2.3. Especificación de Casos de Uso

2.3.1. UC01: Iniciar Partida

- **Descripción:** El jugador inicia una nueva partida o continúa una existente.
- **Flujo Principal:**
 1. El sistema verifica si existe progreso guardado.
 2. Ofrece las opciones “Nueva Partida” o “Continuar”.
 3. El jugador elige.
 4. El sistema carga automáticamente el nivel correspondiente (<<include>> UC09).
 5. Una vez cargado el nivel, se presenta el primer desafío para resolver (<<include>> UC02).
- **Flujos Alternativos:**
 - Si no hay progreso, solo se muestra “Nueva Partida”.
 - Manejo de errores si la base de datos no está disponible.

2.3.2. UC02: Resolver Desafío SQL

- * **Descripción:** El jugador introduce una consulta SQL para superar el desafío del nivel actual.
- * **Precondición:** El nivel debe estar previamente cargado (UC09).
- * **Flujo Principal:**
 1. El sistema presenta automáticamente la narrativa del nivel (<<include>> UC11).
 2. El jugador escribe una consulta SQL.
 3. El sistema valida y ejecuta la consulta automáticamente (<<include>> UC03).
 4. El sistema muestra automáticamente los resultados (<<include>> UC08).
 5. Si la respuesta es correcta, el sistema puede proceder a avanzar de nivel (<<extend>> UC04).
- * **Flujos Alternativos:**
 - * Error de sintaxis: el sistema informa y permite corregir.
 - * Respuesta incorrecta: el sistema informa y permite reintentar.
- **Descripción:** El jugador introduce una consulta SQL para superar el desafío del nivel.
- **Flujo Principal:**

1. El sistema presenta la narrativa y el desafío.
 2. El jugador escribe una consulta SQL.
 3. El sistema valida y ejecuta la consulta (<<include>> UC03).
 4. El sistema muestra los resultados (<<include>> UC08).
 5. Si la respuesta es correcta, se permite avanzar (<<extend>> UC04).
- **Flujos Alternativos:**
 - Error de sintaxis: el sistema informa y permite corregir.
 - Respuesta incorrecta: el sistema informa y permite reintentar.

2.3.3. UC03: Validar Consulta SQL

- **Descripción:** El sistema valida la corrección sintáctica y semántica de la consulta.
- **Flujo Principal:**
 1. Recibe la consulta del jugador.
 2. Verifica la sintaxis y que sea una consulta SELECT.
 3. Valida tablas y columnas referenciadas.
 4. Prepara la consulta para su ejecución.
- **Flujos Alternativos:**
 - Rechaza consultas que no son SELECT (ej. DROP, UPDATE).
 - Informa sobre tablas o columnas inexistentes.

2.3.4. UC04: Avanzar de Nivel

- **Descripción:** El sistema gestiona la transición al siguiente nivel.
- **Flujo Principal:**
 1. Confirma que la respuesta del jugador es correcta.
 2. Actualiza el progreso del jugador.
 3. Guarda el progreso en la base de datos (<<include>> UC05).
 4. Carga el siguiente nivel (<<include>> UC09).
- **Flujos Alternativos:**
 - Si es el último nivel, finaliza el juego.
 - Manejo de errores si no se puede guardar el progreso.

2.3.5. UC05: Guardar Progreso

- **Descripción:** El sistema persiste el estado actual del jugador.
- **Flujo Principal:**
 1. Identifica cambios en el progreso (nivel actual, intentos).
 2. Establece una conexión con la base de datos.
 3. Actualiza el registro del jugador.
- **Flujos Alternativos:**
 - Manejo de errores de conexión o de escritura en la base de datos.



2.3. Especificación de Casos de Uso

2.3.1. UC01: Iniciar Partida

2.3.2. UC02: Resolver Desafío SQL

****Descripción:**** El jugador introduce una consulta SQL para superar el desafío del nivel actual.

****Precondición:**** El nivel debe estar previamente cargado (UC09).

****Flujo Principal:****

1. El sistema presenta automáticamente la narrativa del nivel (<<include>> UC11).
2. El jugador escribe una consulta SQL.
3. El sistema valida y ejecuta la consulta automáticamente (<<include>> UC03).
4. El sistema muestra automáticamente los resultados (<<include>> UC08).
5. Si la respuesta es correcta, el sistema puede proceder a avanzar de nivel (<<extend>> UC04).

****Flujos Alternativos:****

- * Error de sintaxis: el sistema informa y permite corregir.
- * Respuesta incorrecta: el sistema informa y permite reintentar.

2.3.3. UC03: Validar Consulta SQL

****Descripción:**** El sistema valida la corrección sintáctica y semántica de la consulta.

****Actor Principal:**** Sistema de Evaluación

****Flujo Principal:****

1. Recibe la consulta del jugador.
2. Verifica la sintaxis y que sea una consulta SELECT.



3. Valida tablas y columnas referenciadas.
 4. Prepara la consulta para su ejecución.
 5. ****Evalúa automáticamente la respuesta (<<include>> UC10).****
 6. ****Registra automáticamente el intento (<<include>> UC12).****
- * **Flujos Alternativos:****
- * Rechaza consultas que no son SELECT (ej. DROP, UPDATE).
 - * Informa sobre tablas o columnas inexistentes.

2.3.4. UC04: Avanzar de Nivel

- * **Descripción:**** El sistema gestiona la transición al siguiente nivel cuando la respuesta es correcta.
- * **Precondición:**** ****Se ejecuta solo como extensión de UC10 cuando la evaluación es exitosa.****
- * **Flujo Principal:****
1. Confirma que la respuesta del jugador es correcta.
 2. Actualiza el progreso del jugador.
 3. ****Guarda automáticamente el progreso en la base de datos (<<include>> UC05).****
 4. ****Carga automáticamente el siguiente nivel (<<include>> UC09).****
- * **Flujos Alternativos:****
- * Si es el último nivel, finaliza el juego.
 - * Manejo de errores si no se puede guardar el progreso.

2.3.5. UC05: Guardar Progreso

- * **Descripción:**** El sistema persiste el estado actual del jugador.
- * **Precondición:**** ****Se ejecuta automáticamente cuando el jugador avanza de nivel (UC04).****
- * **Flujo Principal:****
1. Identifica cambios en el progreso (nivel actual, intentos).



2. Establece una conexión con la base de datos.

3. Actualiza el registro del jugador.

* **Flujos Alternativos:**

* Manejo de errores de conexión o de escritura en la base de datos.

2.3.6. UC06: Consultar Historial

* **Descripción:** El jugador o docente consulta el historial de partidas y progreso.

* **Actores:** Jugador/Estudiante, Docente

* **Flujo Principal:**

1. El actor solicita consultar el historial.

2. El sistema presenta las opciones de consulta (por jugador, por fecha, etc.).

3. El actor selecciona los criterios de búsqueda.

4. El sistema recupera y muestra la información solicitada.

* **Flujos Alternativos:**

* No hay datos disponibles para mostrar.

* Error de acceso a la base de datos.

2.3.7. UC07: Reiniciar Partida

* **Descripción:** El jugador reinicia su progreso desde el nivel inicial.

* **Flujo Principal:**

1. El jugador solicita reiniciar la partida.

2. El sistema solicita confirmación.

3. El jugador confirma la acción.

4. El sistema resetea el progreso del jugador.

5. **El sistema inicia automáticamente una nueva partida (similar a UC01).**

* **Flujos Alternativos:**

* El jugador cancela la operación.



* Error al resetear los datos.

2.3.8. UC08: Visualizar Resultados

* **Descripción:** El sistema muestra los resultados de la consulta SQL ejecutada.

* **Precondición:** **Se ejecuta automáticamente después de resolver un desafío (UC02).**

* **Flujo Principal:**

1. Recibe los datos resultantes de la consulta validada.
2. Formatea los datos en una tabla legible.
3. Presenta los resultados al jugador.
4. Indica si la respuesta es correcta o incorrecta.

* **Flujos Alternativos:**

- * La consulta no retorna datos (conjunto vacío).
- * Error en la presentación de datos.

2.3.9. UC09: Cargar Nivel

* **Descripción:** El sistema carga la información, narrativa y desafío de un nivel específico.

* **Precondición:** **Se ejecuta automáticamente desde UC01 (Iniciar Partida) o UC04 (Avanzar de Nivel).**

* **Flujo Principal:**

1. Determina qué nivel cargar (inicial o siguiente).
2. Recupera la información del nivel de la base de datos.
3. Prepara las tablas y datos necesarios para el desafío.
4. **Configura el entorno para que el jugador pueda resolver el desafío (<<include>> UC02).**

* **Flujos Alternativos:**

- * El nivel solicitado no existe.
- * Error al acceder a los datos del nivel.



2.3.10. UC10: Evaluar Respuesta

****Descripción:**** El sistema evalúa si la consulta SQL del jugador es correcta.

****Precondición:**** Se ejecuta automáticamente después de validar la consulta (UC03).

****Flujo Principal:****

1. Compara el resultado de la consulta del jugador con la respuesta esperada.
2. Evalúa la eficiencia y calidad de la consulta.
3. Determina si la respuesta es correcta o incorrecta.
4. ****Si es correcta, puede activar el avance de nivel (<<extend>> UC04).****

****Flujos Alternativos:****

- * Respuesta parcialmente correcta (otorga puntos pero no avanza).
- * Múltiples soluciones válidas para el mismo desafío.

2.3.11. UC11: Mostrar Narrativa

****Descripción:**** El sistema presenta la historia y contexto del nivel actual.

****Precondición:**** Se ejecuta automáticamente al resolver un desafío (UC02).

****Flujo Principal:****

1. Recupera el texto narrativo correspondiente al nivel.
2. Presenta la historia de manera atractiva al jugador.
3. Contextualiza el desafío SQL dentro de la narrativa.

****Flujos Alternativos:****

- * Narrativa no disponible para el nivel.
- * Error en la recuperación del contenido.

2.3.12. UC12: Registrar Intento

****Descripción:**** El sistema registra cada intento de resolución para análisis posterior.



* **Precondición:** **Se ejecuta automáticamente después de validar una consulta (UC03).**

* **Flujo Principal:**

1. Captura la consulta SQL introducida por el jugador.
2. Registra timestamp, nivel, y resultado del intento.
3. Almacena la información en la base de datos para análisis.

* **Flujos Alternativos:**

- * Error al escribir en la base de datos.
- * Problema de conectividad durante el registro.



3. Diseño de la Arquitectura Orientada a Objetos

3.1. Diagrama de Clases

El siguiente diagrama de clases modela la estructura del sistema, sus componentes y las relaciones entre ellos.

```
@startuml
!theme plain
title Diagrama de Clases - SQL Escape

package "SQL Escape Game" {

    ' Clase principal del juego
    class SqlEscapeGame {
        - currentPlayer: Player
        - currentLevel: Level
        - gameState: GameState
        - database: GameDatabase
        - sqlEvaluator: SqlEvaluator
        + startGame(): void
        + loadGame(playerId: int): void
        + restartGame(): void
        + getCurrentLevel(): Level
        + processPlayerInput(query: String): QueryResult
        + saveProgress(): boolean
        + exitGame(): void
    }

    ' Gestión de jugadores
    class Player {
        - playerId: int
        - playerName: String
        - currentLevel: int
        - totalScore: int
        - gameStartTime: Date
        - lastPlayTime: Date
        + Player(name: String)
        + getId(): int
        + getName(): String
        + getCurrentLevel(): int
        + setCurrentLevel(level: int): void
        + getScore(): int
        + addScore(points: int): void
        + updateLastPlayTime(): void
        + getPlayTime(): long
    }

}
```



```
' Progreso del jugador
class PlayerProgress {
    - progressId: int
    - playerId: int
    - levelId: int
    - isCompleted: boolean
    - attempts: int
    - completionTime: Date
    - bestQuery: String
+ PlayerProgress(playerId: int, levelId: int)
+ markAsCompleted(): void
+ incrementAttempts(): void
+ setBestQuery(query: String): void
+ getAttempts(): int
+ isLevelCompleted(): boolean
}

' Gestión de niveles
class Level {
    - levelId: int
    - levelNumber: int
    - title: String
    - narrative: String
    - challenge: Challenge
    - isUnlocked: boolean
    - requiredLevel: int
+ Level(id: int, number: int, title: String)
+ getId(): int
+ getNumber(): int
+ getTitle(): String
+ getNarrative(): String
+ getChallenge(): Challenge
+ isAccessible(playerLevel: int): boolean
+ unlock(): void
}

' Desafíos SQL
class Challenge {
    - challengeId: int
    - description: String
    - expectedQuery: String
    - expectedResult: ResultSet
    - hints: List<String>
    - difficulty: DifficultyLevel
    - points: int
+ Challenge(id: int, description: String)
+ getDescription(): String
```




```
+ getExpectedResult(): ResultSet
+ addHint(hint: String): void
+ getHints(): List<String>
+ getPoints(): int
+ getDifficulty(): DifficultyLevel
}

' Evaluador de consultas SQL
class SqlEvaluator {
    - databaseConnection: Connection
    - syntaxValidator: SqlSyntaxValidator
+ SqlEvaluator(connection: Connection)
+ validateQuery(query: String): ValidationResult
+ executeQuery(query: String): QueryResult
+ compareResults(expected: ResultSet, actual: ResultSet):
boolean
    + evaluateChallenge(query: String, challenge: Challenge):
EvaluationResult
    + getSyntaxErrors(query: String): List<String>
}

' Validador de sintaxis
class SqlSyntaxValidator {
    - allowedCommands: Set<String>
+ validateSyntax(query: String): ValidationResult
+ isSelectQuery(query: String): boolean
+ hasValidTables(query: String): boolean
+ hasValidColumns(query: String): boolean
+ checkForbiddenCommands(query: String): List<String>
}

' Resultado de consultas
class QueryResult {
    - resultSet: ResultSet
    - isSuccessful: boolean
    - errorMessage: String
    - executionTime: long
    - rowCount: int
+ QueryResult(resultSet: ResultSet)
+ QueryResult(errorMessage: String)
+ isSuccessful(): boolean
+ getResultSet(): ResultSet
+ getErrorMessage(): String
+ getRowCount(): int
+ getExecutionTime(): long
}
```



```
' Resultado de validación
class ValidationResult {
    - isValid: boolean
    - errors: List<String>
    - warnings: List<String>
    + ValidationResult(isValid: boolean)
    + addError(error: String): void
    + addWarning(warning: String): void
    + getErrors(): List<String>
    + getWarnings(): List<String>
    + hasErrors(): boolean
}

' Resultado de evaluación
class EvaluationResult {
    - isCorrect: boolean
    - score: int
    - feedback: String
    - queryResult: QueryResult
    - validationResult: ValidationResult
    + EvaluationResult(isCorrect: boolean, score: int)
    + getFeedback(): String
    + setFeedback(feedback: String): void
    + getScore(): int
    + isCorrect(): boolean
}

' Base de datos del juego
class GameDatabase {
    - connection: Connection
    - connectionUrl: String
    - username: String
    - password: String
    + GameDatabase(url: String, user: String, pass: String)
    + connect(): boolean
    + disconnect(): void
    + executeQuery(sql: String): ResultSet
    + executeUpdate(sql: String): int
    + getConnection(): Connection
    + isConnected(): boolean
}

' DAO para jugadores
class PlayerDAO {
    - database: GameDatabase
    + PlayerDAO(database: GameDatabase)
    + findById(id: int): Player
}
```



```
+ findByName(name: String): Player
+ save(player: Player): boolean
+ update(player: Player): boolean
+ delete(id: int): boolean
+ getAllPlayers(): List<Player>
}

' DAO para progreso
class ProgressDAO {
    - database: GameDatabase
    + ProgressDAO(database: GameDatabase)
    + findByPlayer(playerId: int): List<PlayerProgress>
    + findByLevel(levelId: int): List<PlayerProgress>
    + save(progress: PlayerProgress): boolean
    + update(progress: PlayerProgress): boolean
}
}
@enduml
```



3.2. Descripción de Clases Principales

3.2.1. *SqlEscapeGame*

- **Responsabilidad:** Es la clase orquestadora principal. Controla el flujo del juego, gestiona el estado actual y coordina las interacciones entre el jugador, los niveles y el sistema de evaluación.

3.2.2. *Player y PlayerProgress*

- **Player:** Modela al jugador, almacenando su información básica como nombre, nivel actual y puntuación.
- **PlayerProgress:** Registra información detallada del progreso del jugador en cada nivel, como los intentos, si fue completado y la mejor consulta realizada.

3.2.3. *Level y Challenge*

- **Level:** Representa un nivel del juego. Contiene la narrativa, el título y el desafío asociado.
- **Challenge:** Encapsula la lógica del desafío SQL, incluyendo la descripción, la consulta esperada (o una forma de validarla) y las pistas.

3.2.4. *SqlEvaluator y SqlSyntaxValidator*

- **SqlEvaluator:** Se encarga de ejecutar la consulta del jugador y comparar el resultado con la solución esperada para determinar si es correcta.
- **SqlSyntaxValidator:** Realiza una validación previa de la consulta para asegurar que es sintácticamente correcta y que no contiene comandos prohibidos.

3.2.5. *GameDatabase y Clases DAO*

- **GameDatabase:** Gestiona la conexión con la base de datos MySQL. Proporciona métodos para ejecutar consultas y actualizaciones.
- **Clases DAO (PlayerDAO, ProgressDAO):** Implementan el patrón *Data Access Object*. Abstraen la lógica de acceso a datos, separando la lógica de negocio de la de persistencia.

3.3. Patrones de Diseño Aplicados

- **Singleton:** La clase *GameDatabase* podría implementarse como un Singleton para asegurar una única instancia de conexión a la base de datos en toda la aplicación.
- **Data Access Object (DAO):** Utilizado para encapsular el acceso a la base de datos, mejorando la modularidad y facilitando el mantenimiento.
- **State:** El estado del juego (*gameState*) puede ser manejado con el patrón State para gestionar transiciones complejas (ej. *Playing*, *Paused*, *Finished*).
- Facade ver



- MVC ver

2 Descripción general

2.1 Perspectiva del producto

SQL Escape es una aplicación autónoma desarrollada en el contexto de las unidades curriculares Programación II y Base de Datos. Se trata de un juego educativo con fines formativos, que combina narrativa interactiva, resolución de acertijos y consultas SQL como mecánica principal de avance. El producto está diseñado como una solución original y autoconclusiva, sin dependencia de otros sistemas externos. Internamente, integra un motor de juego desarrollado en Java, una base de datos relacional para almacenamiento persistente y una interfaz de usuario textual o gráfica según la elección del grupo. El diseño sigue principios de programación orientada a objetos y hace uso de patrones de diseño adecuados para garantizar modularidad y mantenibilidad.

2.2 Funcionalidad del producto

[Inserte aquí el texto]

Resumen de las funcionalidades principales que el producto debe realizar, sin entrar en información de detalle.

En ocasiones la información de esta sección puede tomarse de un documento de especificación del sistema de mayor nivel (ej. Requisitos del sistema).

Las funcionalidades deben estar organizadas de manera que el cliente o cualquier interlocutor pueda entenderlo perfectamente. Para ello se pueden utilizar métodos textuales o gráficos.

2.3 Restricciones

En esta versión no se contará con opción de multijugador, tampoco contará con una versión



mobile.

Será un proyecto local, con una base de datos propia.

No se dispone de un sistema de ranking.

2.4 Suposiciones y dependencias

[Inserte aquí el texto]

Descripción de aquellos factores que, si cambian, pueden afectar a los requisitos. Por ejemplo una asunción puede ser que determinado sistema operativo está disponible para el hardware requerido. De hecho, si el sistema operativo no estuviera disponible, la SRS debería modificarse.

2.5 Evolución previsible del sistema

[Inserte aquí el texto]

Identificación de futuras mejoras al sistema, que podrán analizarse e implementarse en un futuro.



2. Análisis de Requisitos y Casos de Uso

2.1. Actores del Sistema

2.1.1. Jugador/Estudiante

Descripción: Es el usuario final que interactúa directamente con el juego. Su objetivo es progresar en la historia resolviendo los desafíos SQL.

2.1.2. Docente

Descripción: Un actor secundario que puede utilizar el juego como una herramienta pedagógica. Podría tener acceso a los resultados y al progreso de los estudiantes para evaluar su desempeño.

2.1.3. Sistema de Evaluación

Descripción: Un componente interno y automatizado del software. Es responsable de validar las consultas SQL, ejecutar las pruebas y determinar si las respuestas del jugador son correctas.

2.2. Diagrama de Casos de Uso

A continuación, se presenta el diagrama de casos de uso que ilustra las interacciones entre los actores y el sistema.

3 Requisitos específicos

Esta es la sección más extensa y más importante del documento.

Debe contener una lista detallada y completa de los requisitos que debe cumplir el sistema a desarrollar. El nivel de detalle de los requisitos debe ser el suficiente para que el equipo de desarrollo pueda diseñar un sistema que satisfaga los requisitos y los encargados de las pruebas puedan determinar si éstos se satisfacen.

Los requisitos se dispondrán en forma de listas numeradas para su identificación, seguimiento, trazabilidad y validación (ej. RF 10, RF 10.1, RF 10.2,...).

Para cada requisito debe completarse la siguiente tabla:

Ref*	Nombre	Tipo	Prioridad
RF01	Validación de consultas SQL	Funcional	Alta
RF02	Evaluación de respuestas	Funcional	Alta
RF03	Visualización de resultados	Funcional	Alta
RF04	Carga progresiva de niveles	Funcional	Alta
RF05	Persistencia del progreso	Funcional	Alta
RF06	Registro de intentos	Funcional	Media
RF07	Control de finalización del juego	Funcional	Media
RF08	Reinicio de partida	Funcional	Media
RF09	Presentación narrativa con imágenes y pregunta	Funcional	Alta
RF10	Consulta SQL por nivel	Funcional	Alta

**Ref= Referencia*

Nombre: Máximo 3 palabras

Tipo:

La distribución de los párrafos que forman este punto puede diferir del propuesto en esta plantilla, si las características del sistema aconsejan otra distribución para ofrecer mayor claridad en la exposición.

Número de requisito	[Inserte aquí el texto]
Nombre de requisito	[Inserte aquí el texto]
Tipo	Requisito Restricción
Fuente del requisito	[Inserte aquí el texto]
Prioridad del requisito	Alta/Esencial Media/Deseado Baja/ Opcional

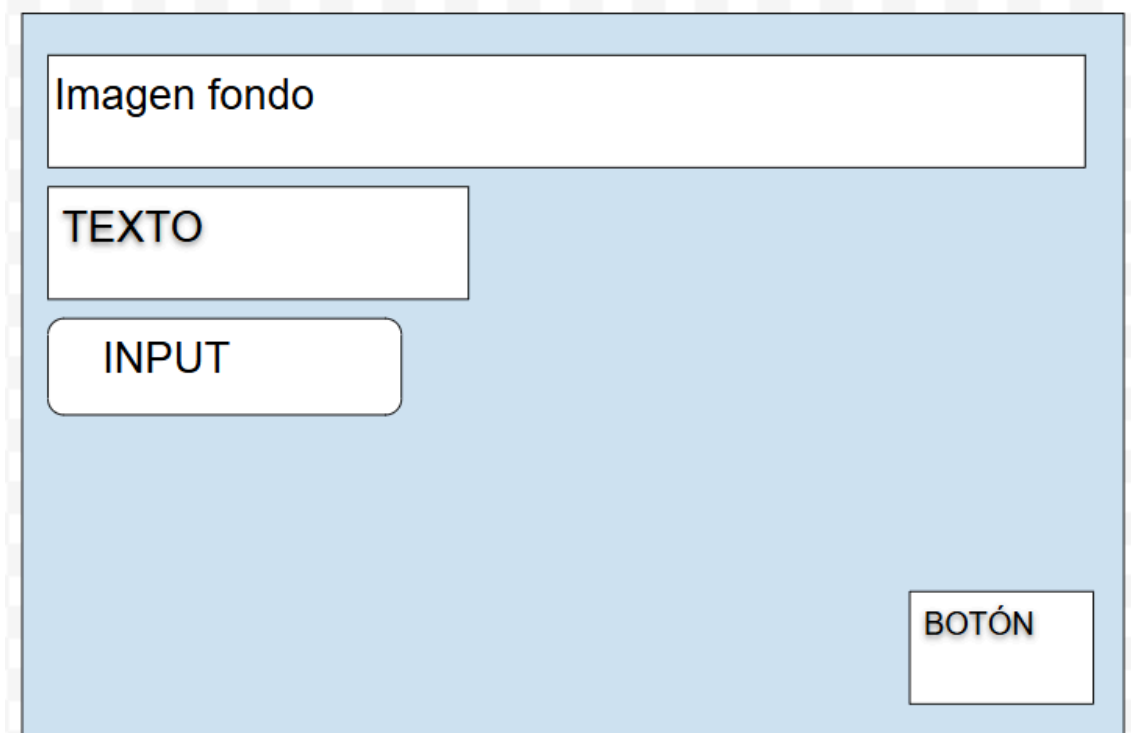
3.1 Requisitos comunes de los interfaces

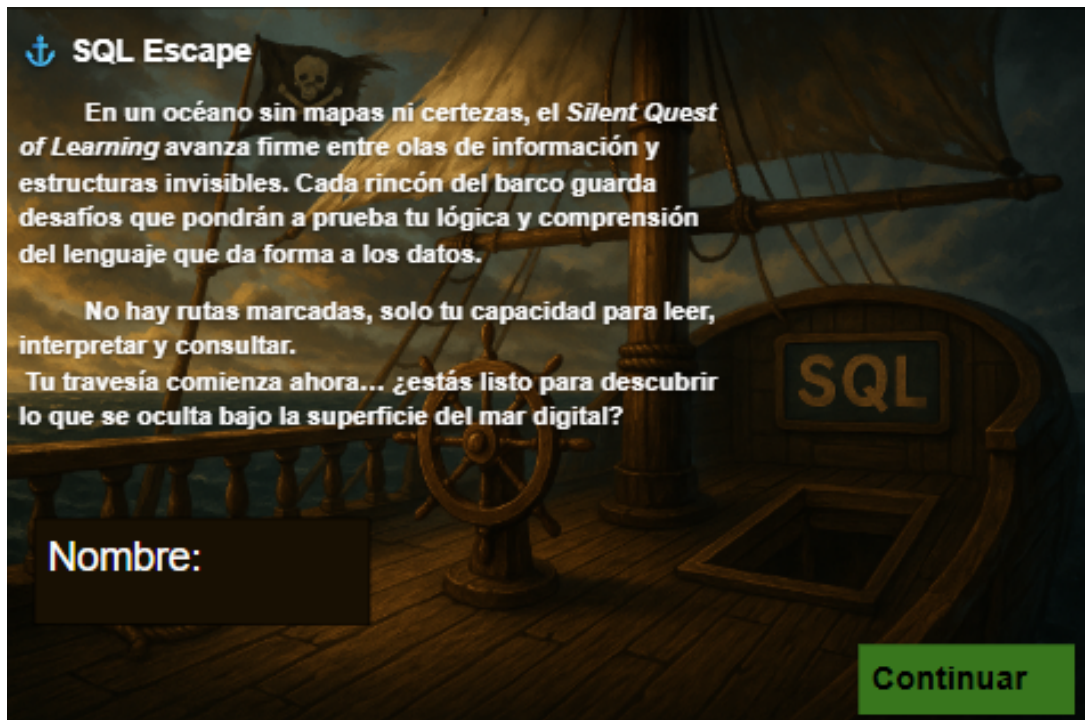
[Inserte aquí el texto]

Descripción detallada de todas las entradas y salidas del sistema de software.

3.1.1 Interfaces de usuario

El usuario al ingresar al juego se encontrará con una ventana de texto donde se presentará el juego para luego toparse con una ventana de respuesta para escribir un "Nickname", seguido de un botón "Comenzar" o "Jugar" para dar inicio al juego.





CONTEXTTO Y PROBLEMA:

texto

IMAGEN

Input

BOTÓN

Texto

BOTÓN

3.1.4 Interfaces de comunicación

[Inserte aquí el texto]

Describir los requisitos del interfaces de comunicación si hay comunicaciones con otros sistemas y cuales son las protocolos de comunicación.

3.2 Requisitos funcionales

[Inserte aquí el texto]

Definición de acciones fundamentales que debe realizar el software al recibir información, procesarla y producir resultados.

En ellas se incluye:

- *Comprobación de validez de las entradas*
- *Secuencia exacta de operaciones*
- *Respuesta a situaciones anormales (desbordamientos, comunicaciones, recuperación de errores)*
- *Parámetros*
- *Generación de salidas*
- *Relaciones entre entradas y salidas (secuencias de entradas y salidas, formulas para la conversión de información)*
- *Especificación de los requisitos lógicos para la información que será almacenada en base de datos (tipo de información, requerido)*

Las requisitos funcionales pueden ser divididos en sub-secciones.

3.2.1 Requisito funcional 1

3.2.2 Requisito funcional 2

3.2.3 Requisito funcional 3

3.2.4 Requisito funcional n

3.3 Requisitos no funcionales

3.3.1 Requisitos de rendimiento

[Inserte aquí el texto]

Especificación de los requisitos relacionados con la carga que se espera tenga que soportar el sistema. Por ejemplo, el número de terminales, el número esperado de usuarios simultáneamente conectados, número de transacciones por segundo que deberá soportar el sistema, etc.

Todos estos requisitos deben ser medibles. Por ejemplo, indicando “el 95% de las transacciones deben realizarse en menos de 1 segundo”, en lugar de “los operadores no deben esperar a que se complete la transacción”.

3.3.2 Seguridad

[Inserte aquí el texto]

Especificación de elementos que protegerán al software de accesos, usos y sabotajes maliciosos, así como de modificaciones o destrucciones maliciosas o accidentales. Los requisitos pueden especificar:

- *Empleo de técnicas criptográficas.*
- *Registro de ficheros con “logs” de actividad.*
- *Asignación de determinadas funcionalidades a determinados módulos. ▪ Restricciones de comunicación entre determinados módulos.*
- *Comprobaciones de integridad de información crítica.*



3.3.3 Fiabilidad

[Inserte aquí el texto]

Especificación de los factores de fiabilidad necesaria del sistema. Esto se expresa generalmente como el tiempo entre los incidentes permisibles, o el total de incidentes permisible.

3.3.4 Disponibilidad

[Inserte aquí el texto]

Especificación de los factores de disponibilidad final exigidos al sistema. Normalmente expresados en % de tiempo en los que el software tiene que mostrar disponibilidad.

3.3.5 Mantenibilidad

[Inserte aquí el texto]

Identificación del tipo de mantenimiento necesario del sistema.

Especificación de quien debe realizar las tareas de mantenimiento, por ejemplo usuarios, o un desarrollador.

Especificación de cuando debe realizarse las tareas de mantenimiento. Por ejemplo, generación de estadísticas de acceso semanales y mensuales.

3.3.6 Portabilidad

[Inserte aquí el texto]

Especificación de atributos que debe presentar el software para facilitar su traslado a otras plataformas u entornos. Pueden incluirse:

- *Porcentaje de componentes dependientes del servidor.*
- *Porcentaje de código dependiente del servidor.*
- *Uso de un determinado lenguaje por su portabilidad.*
- *Uso de un determinado compilador o plataforma de desarrollo.*
- *Uso de un determinado sistema operativo.*



4 Análisis

Esta sección presenta el análisis técnico del sistema *SQL Escape*, centrado en dos dimensiones complementarias: la estructura de la base de datos y la lógica funcional del programa. El objetivo es establecer las bases conceptuales necesarias para pasar a la etapa de diseño, garantizando una comprensión clara de los componentes del sistema, su interacción y los requerimientos técnicos involucrados.

4.1 Análisis de base de datos

En esta subsección se desarrollará el modelo de datos que sustentará la lógica del juego. Se elaborará un **Modelo Entidad-Relación (MER)** que refleje las entidades principales del universo narrativo y sus relaciones, el cual se transformará luego en un **Modelo Relacional (MR)** compatible con sistemas de gestión de bases de datos relacionales. A partir de este modelo, se redactarán los scripts SQL necesarios, organizados por tipo de lenguaje:

- **DDL (Data Definition Language)**: creación de tablas y estructuras;
- **DML (Data Manipulation Language)**: inserción, modificación y eliminación de datos;
- **DCL (Data Control Language)**: asignación de privilegios;
- **TCL (Transaction Control Language)**: control de transacciones (si corresponde).

4.1.1 Modelo Entidad-Relación (MER)

Para el desarrollo del videojuego *SQL Escape* se diseñan **dos modelos entidad-relación complementarios**, que cumplen funciones distintas pero integradas dentro del sistema.

a) Modelo Entidad-Relación del universo del juego (base de datos de los desafíos)


El primer modelo representa el universo narrativo sobre el cual se ejecutan las consultas SQL de cada nivel. Esta base de datos es el eje central de los desafíos, ya que cada nivel está construido sobre preguntas contextualizadas en este mundo ficticio.

Entidades y relaciones principales:

- **MARINERO**: representa a cada miembro de la tripulación. Se almacena su nombre, edad, descripción y país de origen.



- **BARCO**: entidad clave que describe a las embarcaciones disponibles en el universo, incluyendo características técnicas y de construcción.
- **TRIPULANTE** (relación): relación muchos a muchos entre MARINERO y BARCO. Incluye atributos propios como **rango**, **fecha_cese** y **años_servicio**.
- **NAVEGACIÓN** (relación): vincula a cada BARCO con una ISLA, registrando la fecha y el estado de la navegación.
- **ISLA**: entidad geográfica que describe cada destino posible. Se registra su nombre, habitantes, flora, fauna y estructura.
- **SENSOR**: sensores instalados en los barcos. Se almacenan por tipo, ubicación y años de servicio.
- **REGISTRO_COMBUSTIBLE**: vinculado a BARCO, registra los consumos de combustible por fecha e identificación de carga.

 *Este modelo será utilizado como base para la construcción de los desafíos en cada nivel, permitiendo escribir consultas SELECT, JOIN, filtros, agrupamientos, subconsultas, etc.*

b) Modelo Entidad-Relación del juego (gestión de usuarios, niveles y progreso)

El segundo modelo, que será desarrollado a continuación, representará los elementos internos del juego: jugadores/as, niveles, consultas realizadas y progresos individuales. Este modelo permite:

- Registrar a cada jugador/a por nickname.
- Asociar el progreso por niveles.
- Almacenar cada intento de consulta, su validez y resultado.
- Controlar el avance, reinicio o finalización del juego.

[Espacio reservado para el futuro MER del sistema de juego y progreso]

4.1.2 Modelo Relacional (MR)

Tablas Relacionales

MARINERO(id, nombre, edad, descripcion, pais_origen)
BARCO(nro_barco, camarotes, max_combustible, pais_origen, material, velocidad_crucero)
ISLA(nombre, flora, habitantes, fauna, puerto, estructura)
SENSOR(nro_barco, tipo, ubicacion, anios_servicio)
REGISTRO_COMBUSTIBLE(nro_barco, fecha, cantidad_utilizada)
TRIPULANTE(id, nro_barco, fecha_ingreso, fecha_cese, rango)
NAVEGACION(nro_barco, nombre, fecha, estado)

PI (Dependencias de Inclusión)

$\Pi_{id}(TRIPULANTE) \subseteq \Pi_{id}(MARINERO)$
 $\Pi_{nro_barco}(TRIPULANTE) \subseteq \Pi_{nro_barco}(BARCO)$
 $\Pi_{nro_barco}(NAVEGACION) \subseteq \Pi_{nro_barco}(BARCO)$
 $\Pi_{nombre}(NAVEGACION) \subseteq \Pi_{nombre}(ISLA)$
 $\Pi_{nro_barco}(SENSOR) \subseteq \Pi_{nro_barco}(BARCO)$
 $\Pi_{nro_barco}(REGISTRO_COMBUSTIBLE) \subseteq \Pi_{nro_barco}(BARCO)$

Restricciones:

- {Un barco no puede tener más de un mismo estado en la misma fecha}
- {Los estados posibles en navegación son entrante y saliente}

4.1.2.1 Análisis de Normalización

Con el objetivo de asegurar la integridad de los datos y evitar redundancias o anomalías de actualización, se aplicó el proceso de normalización al modelo relacional derivado del MER del universo del juego. A continuación, se detalla el análisis de cumplimiento de las tres primeras formas normales (1FN, 2FN y 3FN) para cada una de las tablas del modelo.

BARCO(nro_barco, camarotes, max_combustible, pais_origen, material, velocidad_crucero)

- Cumple 1FN: todos los atributos son atómicos.
- Cumple 2FN: todos los atributos dependen completamente de la clave primaria (nro_barco).



- Cumple 3FN: no existen dependencias transitivas entre atributos no clave.

MARINERO(id, nombre, edad, descripcion, pais_origen)

- Cumple 1FN: los atributos son indivisibles.
- Cumple 2FN: todos los atributos dependen funcionalmente de la clave primaria (**id**).
- Cumple 3FN: no hay dependencias entre atributos no clave.

ISLA(nombre, flora, habitantes, fauna, puerto, estructura)

- Cumple 1FN: todos los campos contienen valores simples.
- Cumple 2FN: los atributos dependen totalmente de la clave primaria (**nombre**).
- Cumple 3FN: no se identifican dependencias transitivas.

REGISTRO_COMBUSTIBLE(nro_barco, fecha, cantidad_utilizada)

- Se considera como clave primaria compuesta (**nro_barco, fecha**).
- Cumple 1FN: valores atómicos.
- Cumple 2FN: **cantidad_utilizada** depende del conjunto completo de la clave.
- Cumple 3FN: no hay atributos no clave dependientes de otros atributos no clave.

TRIPULANTE(id, nro_barco, fecha_ingreso, fecha_cese, rango)

- Clave primaria compuesta (**id, nro_barco**).
- Cumple 1FN: todos los campos son atómicos.
- Cumple 2FN: los atributos adicionales dependen de toda la clave compuesta.
- Cumple 3FN: no existen dependencias entre atributos no clave.

NAVEGACION(nro_barco, nombre, fecha, estado)



- Clave primaria compuesta (**nro_barco**, **nombre**, **fecha**).
- Cumple 1FN: estructura sin repeticiones ni multivalores.
- Cumple 2FN: **estado** depende de la combinación completa.
- Cumple 3FN: no hay dependencias transitivas.

SENSOR(nro_barco, tipo, ubicacion, anios_servicio)

- Se asume clave primaria compuesta (**nro_barco**, **tipo**) si hay múltiples sensores por barco.
- Cumple 1FN: valores simples.
- Cumple 2FN: los atributos dependen del par clave.
- Cumple 3FN: no existen dependencias entre atributos no clave.

En conclusión, todas las tablas del modelo se encuentran **normalizadas hasta la Tercera Forma Normal (3FN)**. Esto garantiza un modelo robusto, sin redundancias innecesarias, que facilita tanto la integridad como el mantenimiento de los datos en el contexto del juego SQL Escape.



4.1.2.2 Álgebra Relacional

Con el objetivo de validar la coherencia del modelo relacional y ejemplificar las operaciones que servirán como base para los desafíos del juego, se presenta una serie de expresiones en **álgebra relacional** aplicadas sobre las tablas normalizadas del universo del juego.

Estas expresiones permiten anticipar los tipos de consultas que realizarán los/as jugadores/as en los distintos niveles, y constituyen una herramienta formal para analizar la consistencia del esquema de base de datos.

a) Proyección de nombres de marineros uruguayos

plaintext

CopiarEditar

```
 $\pi(\text{nombre})(\sigma(\text{pais\_origen} = \text{'Uruguay'})(\text{MARINERO}))$ 
```

 Recupera los nombres de todos los marineros cuyo país de origen es Uruguay.

b) Listar barcos con más de 100 camarotes

plaintext

CopiarEditar

```
 $\sigma(\text{camarotes} > 100)(\text{BARCO})$ 
```


 Selecciona los registros de barcos cuya capacidad supera los 100 camarotes.

c) Nombres de marineros que integran un barco

plaintext

CopiarEditar

```
 $\pi(\text{nombre})($   
     $\text{MARINERO} \bowtie \text{TRIPULANTE}$   
)
```

 Realiza una combinación natural entre **MARINERO** y **TRIPULANTE** para obtener los nombres de marineros asignados a algún barco.


d) Información de barcos que han navegado hacia islas con puerto

plaintext

CopiarEditar



```
 $\pi(\text{BARCO.nro\_barco}, \text{nombre}, \text{fecha})($   
   $(\text{NAVEGACION} \bowtie \text{ISLA}) \bowtie \sigma(\text{puerto} = \text{true})(\text{ISLA})$   
)
```


 A través de una combinación múltiple, se extrae la navegación de barcos hacia islas que poseen puerto.

e) Barcos con sensores en cubierta

plaintext

CopiarEditar

```
 $\pi(\text{nro\_barco})($   
   $\sigma(\text{ubicacion} = \text{'cubierta'})(\text{SENSOR})$   
)
```

 Selecciona los identificadores de los barcos que tienen sensores ubicados en la cubierta.

Estas expresiones muestran una diversidad de operadores del álgebra relacional: **proyección (π)**, **selección (σ)** y **combinación (\bowtie)**, que constituyen la base lógica para las consultas SQL a ser formuladas por los/as jugadores/as en los niveles del juego.

4.1.3 DDL (*Data Definition Language*)

En esta sección se definen las instrucciones SQL necesarias para la **creación de la base de datos** que sustenta los desafíos del juego. Se utiliza el lenguaje de definición de datos (DDL) para declarar las estructuras principales: tablas, claves primarias, claves foráneas y restricciones básicas.

El siguiente script puede ser ejecutado en sistemas de gestión de bases de datos relacionales compatibles con SQL estándar, como PostgreSQL o MySQL (con mínimas adaptaciones de tipos).

```
CREATE DATABASE IF NOT EXISTS sqlescape;  
USE sqlescape;
```



```
CREATE TABLE MARINERO (  
  id INT PRIMARY KEY AUTO_INCREMENT,  
  nombre VARCHAR(50) NOT NULL,  
  edad INT,  
  descripcion TEXT,  
  pais_origen VARCHAR(50)  
);
```

```
CREATE TABLE BARCO (  
  nro_barco INT PRIMARY KEY AUTO_INCREMENT,  
  camarotes INT,  
  max_combustible DECIMAL(10,2),  
  pais_origen VARCHAR(50),  
  material VARCHAR(50),  
  velocidad_crucero DECIMAL(5,2)  
);
```

```
CREATE TABLE ISLA (  
  nombre VARCHAR(50) PRIMARY KEY,  
  flora TEXT,  
  habitantes INT,  
  fauna TEXT,  
  puerto BOOLEAN,  
  estructura TEXT  
);
```

```
CREATE TABLE SENSOR (  
  nro_barco INT,  
  tipo VARCHAR(30),  
  ubicacion VARCHAR(50),  
  anios_servicio INT,  
  PRIMARY KEY (nro_barco, tipo),  
  FOREIGN KEY (nro_barco) REFERENCES BARCO(nro_barco)  
);
```

```
CREATE TABLE REGISTRO_COMBUSTIBLE (  
  nro_barco INT,  
  fecha DATE,  
  cantidad_utilizada DECIMAL(10,2),  
  PRIMARY KEY (nro_barco, fecha),  
  FOREIGN KEY (nro_barco) REFERENCES BARCO(nro_barco)  
);
```

```
CREATE TABLE TRIPULANTE (  
  id INT,  
  nro_barco INT,  
  fecha_ingreso DATE,
```



```
    fecha_cese DATE,  
    rango VARCHAR(30),  
    PRIMARY KEY (id, nro_barco),  
    FOREIGN KEY (id) REFERENCES MARINERO(id),  
    FOREIGN KEY (nro_barco) REFERENCES BARCO(nro_barco)  
);  
  
CREATE TABLE NAVEGACION (  
    nro_barco INT,  
    nombre VARCHAR(50),  
    fecha DATE,  
    estado ENUM('entrante', 'saliente'),  
    PRIMARY KEY (nro_barco, nombre, fecha),  
    FOREIGN KEY (nro_barco) REFERENCES BARCO(nro_barco),  
    FOREIGN KEY (nombre) REFERENCES ISLA(nombre)  
);
```

4.1.4 DML (Data Manipulation Language)

El sistema utilizará sentencias DML para gestionar la información almacenada en la base de datos. Estas sentencias permitirán realizar operaciones como inserción, actualización, eliminación y consulta de datos. A continuación se describen las principales operaciones DML que serán utilizadas:

SELECT: Para consultar y recuperar información específica desde las tablas, como productos disponibles, usuarios registrados, historial de pedidos, etc.

INSERT: Para agregar nuevos registros a las tablas, como por ejemplo registrar un nuevo usuario, agregar un nuevo producto o registrar una nueva venta.

UPDATE: Para modificar datos existentes, por ejemplo actualizar el stock de un producto o la información de un usuario.

DELETE: Para eliminar registros obsoletos o innecesarios, como productos discontinuos o usuarios inactivos.



4.1.5 DCL (Data Control Language)

En este proyecto no se utilizarán sentencias DCL (Data Control Language), ya que el control de accesos y roles se gestionará directamente desde la lógica del sistema desarrollado en Java. Los permisos de administrador y usuario se manejarán mediante validaciones internas y no a nivel de base de datos. Esta decisión simplifica la administración del sistema. Por lo tanto, no será necesario aplicar GRANT ni REVOKE en la base de datos.

Las principales sentencias DCL:

GRANT: Para conceder permisos específicos a los usuarios o roles, como lectura, inserción o modificación de datos. Por ejemplo, permitir que un administrador tenga acceso completo a todas las tablas, mientras que un usuario común solo puede consultar ciertos datos.

REVOKE: Para retirar permisos previamente otorgados a usuarios o roles, en caso de que sus responsabilidades cambien o su acceso ya no sea necesario.

4.1.6 TCL (Transaction Control Language)

En este proyecto no se utilizarán sentencias TCL (Transaction Control Language) de forma explícita, ya que el control de transacciones será manejado por el motor de base de datos y la lógica del sistema Java. Operaciones como COMMIT o ROLLBACK se gestionarán automáticamente al ejecutar las consultas desde el código(JAVA). Esta decisión busca simplificar el desarrollo. No será necesario un control manual de transacciones en SQL.

4.2 Análisis de programación

Aquí se identifican y describen los elementos funcionales del sistema desde una perspectiva lógica, sin llegar aún al nivel gráfico o estructural del diseño. Se detallarán los casos de uso, los actores involucrados, los escenarios narrativos por nivel, y se construirá un glosario de términos técnicos y narrativos que servirá de referencia en las siguientes fases. Los diagramas UML, incluyendo los de clases, actividades o secuencia, se reservarán para la sección de diseño, ya que allí pasarán a representar formalmente la arquitectura del sistema y su implementación en Java.



5 Diseño



6 Implementación



7 Apéndices

[Inserte aquí el texto]

Pueden contener todo tipo de información relevante para la SRS pero que, propiamente, no forme parte de la SRS.