Manual Tecnico

MythQL

Fecha: ***06/11/2025***

Cliente: ***Pablo Linares***

Integrantes del Proyecto: ***Gabriel Denuble***

Benjamin Moyano

Martin Blacha

Franco Carnevali

1 - Objeto/Propósito

Este documento tiene como propósito, poder profundizar el funcionamiento del sistema en sí, junto a sus entidades que comparten papel, sus características y relaciones. El lenguaje y funcionalidad del sistema se encontrará en un documento aparte, en donde este mismo será ubicado al final de este documento junto a otros necesarios.

1.1 - Alcance

Este documento, también, describe los componentes internos del sistema, los procesos de fondo que permiten la ejecución de consultas

1.2 - Introducción General a MythQL

MythQL es un lenguaje y gestor de base de datos que tendrá su propia sintaxis pero basado en los modelos de base de datos relacionales.

Se tomará como referencia al enunciado de la tarea final “Trabajo Grupal - Programación Sobre Redes - 2025.pdf” (Se acoplara al glosario)

y se tomará como base la sintaxis que tiene el MySQL WORKBENCH, pero la forma de usar los comandos serán distintas, aparte de cambiar el nombre de los comandos de cada comando del MySQL se cambiará la forma de la gramática.

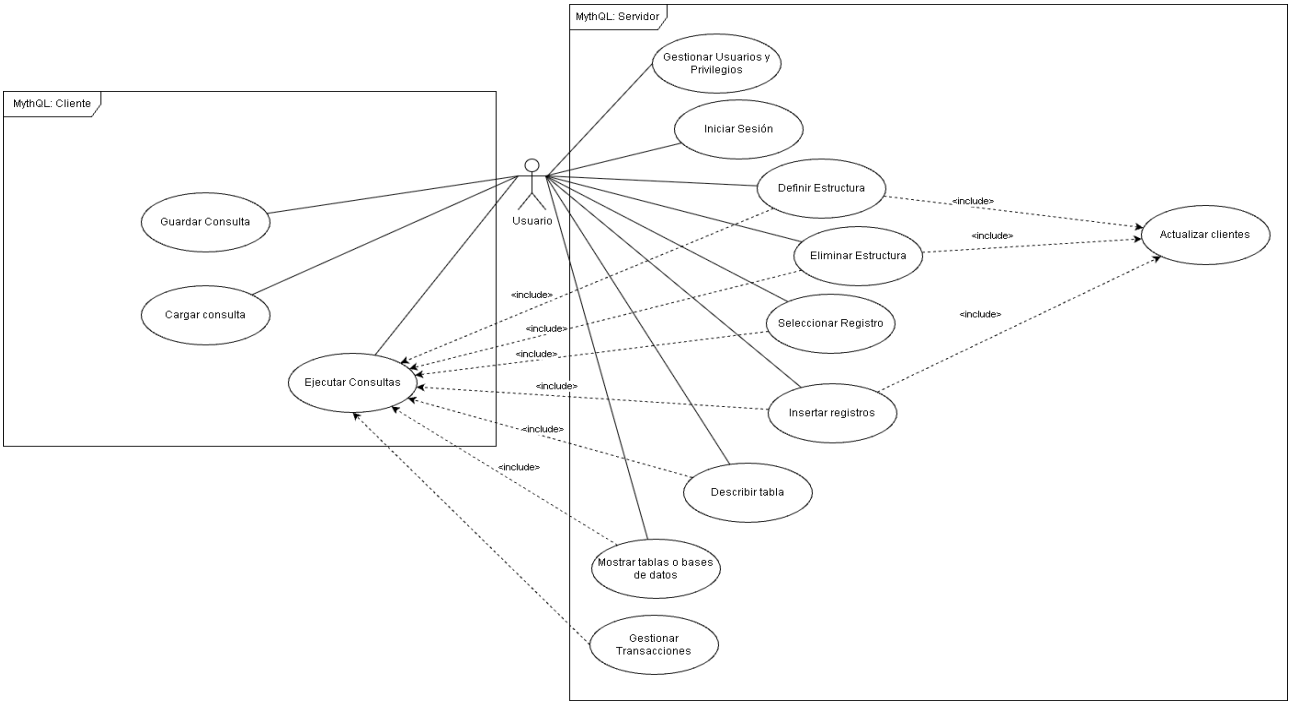
También se basará en parte en Oracle, por que tiene que iniciar sesión, la persona que se crea la cuenta para usar el gestor de base de datos podrán crear otros usuarios internos en el gestor de base de datos.

Estos Usuarios creados de forma interna podrá iniciar sesion como si fuera la cuenta principal, tendrá privilegios delimitados o establecidos por la cuenta principal(usted)

1.3 - Mapas del Sistema/Funcionalidad

En esta sección se describen los diagramas/vistas que toman juego a entender la funcionalidad del sistema de una forma teórica

1.3.1 - Vista de Caso de Uso



1.3.1.1 - Usuario: El sistema MythQL se estructura en torno a un único actor principal: el **Usuario**, quien representa a cualquier persona que interactúa con el gestor de base de datos para realizar operaciones de creación, modificación, eliminación y consulta de información mediante el lenguaje de comandos propietario del sistema. Este actor posee tres atributos fundamentales: un **Nombre** (identificador alfanumérico que puede ser generado automáticamente o definido manualmente), un **Rol** (clasificación alfabética que define privilegios como admin, scribe, seer o guard), y una **Contraseña** (cadena alfanumérica con caracteres especiales para autenticación segura). Todos los casos de uso del sistema requieren la participación activa de este actor, quien ejecuta comandos y recibe respuestas del gestor de base de datos.

1.3.1.2 - Iniciar Sesion:

Este caso de uso representa el punto de entrada obligatorio al sistema MythQL, donde el usuario debe autenticarse y establecer una sesión segura con el servidor de base de datos.

El proceso comienza cuando el sistema presenta un diálogo de configuración donde el usuario ingresa la dirección IP y puerto del servidor, seguido de una validación de conectividad mediante prueba de conexión.

Una vez establecida la conexión, el sistema solicita las credenciales de acceso (usuario y contraseña) que son enviadas al servidor mediante el comando LOGIN.

El servidor realiza múltiples validaciones: verifica que las credenciales sean correctas contra el archivo de usuarios, confirma que el usuario no tenga sesiones activas en otras conexiones para evitar duplicados, y posteriormente genera un token único de sesión que se almacena en el mapa de sesiones activas del servidor. El cliente recibe este token de autenticación y procede a abrir la interfaz principal del sistema, estableciendo además una conexión dedicada para recibir notificaciones en tiempo real sobre cambios en el sistema.

Este caso de uso es prerrequisito absoluto para todos los demás casos funcionales del sistema, ya que sin una sesión activa válida ninguna otra operación puede ejecutarse.

1.3.1.3 - Ejecutar Consultas: Este caso de uso proporciona el mecanismo central de interacción con el sistema, permitiendo procesar y ejecutar comandos escritos en lenguaje MythQL.

El usuario redacta comandos utilizando la sintaxis específica de MythQL en un editor que proporciona resaltado de sintaxis automático para facilitar la identificación de palabras clave y estructuras del lenguaje.

Una vez escrito el comando, el usuario selecciona el modo de ejecución según sus necesidades: puede ejecutar la consulta completa, solamente el texto que ha seleccionado, o únicamente la línea actual donde se encuentra el cursor.

El sistema valida exhaustivamente la sintaxis del comando antes de enviarlo al servidor para su procesamiento, verificando que la estructura corresponda con las reglas gramaticales de MythQL.

Si la validación es exitosa, el comando se transmite al servidor que lo procesa y retorna una respuesta.

Finalmente, el sistema muestra los resultados en una consola con formato diferenciado por colores para mejorar la legibilidad, distinguiendo entre datos, mensajes de sistema, errores y confirmaciones.

Este caso de uso depende de RF-01 como precondición y actúa como base fundamental para RF-03, RF-04, RF-05, RF-06, RF-07, RF-08 y RF-09, ya que todos estos casos utilizan este mecanismo para ejecutar sus comandos específicos.

1.3.1.4 - Definir Estructura: Este caso de uso permite al usuario crear nuevas bases de datos o tablas en el sistema mediante el comando SUMMON, estableciendo la estructura fundamental del sistema de información.

El usuario escribe un comando SUMMON especificando si creará una base de datos completa o una tabla con sus respectivas columnas y tipos de datos, y procede a ejecutar el script.

El sistema verifica exhaustivamente la sintaxis del comando, y en caso de tratarse de una tabla, realiza validaciones adicionales: confirma que exista una base de datos seleccionada actualmente (ya que las tablas deben pertenecer a una base de datos) y verifica que los tipos de datos especificados para cada columna sean válidos según las definiciones del sistema (INT, VARCHAR, etc.).

Una vez superadas todas las validaciones, la solicitud se envía al servidor para procesamiento, donde se ejecuta la creación física: para bases de datos se generan los archivos necesarios en el sistema de archivos del servidor, mientras que para tablas se agrega la definición estructural al archivo de metadatos de la base de datos activa.

El sistema confirma la creación exitosa mediante un mensaje al usuario, notifica a todos los usuarios conectados sobre este cambio estructural para mantener consistencia, actualiza la lista de estructuras disponibles en la interfaz, y finalmente ejecuta automáticamente RF-11 (Actualizar Cliente) para sincronizar las vistas de todos los clientes.

Este caso de uso requiere RF-01 (Iniciar Sesión) como precondición, utiliza RF-02 (Ejecutar Consultas) como mecanismo de ejecución, y las estructuras que crea son utilizadas posteriormente por RF-05, RF-06, RF-08 y RF-09.

1.3.1.5 - Eliminar Estructura: Este caso de uso proporciona la funcionalidad de destrucción de estructuras de datos, permitiendo al usuario eliminar bases de datos o tablas existentes mediante el comando BURN.

El proceso inicia cuando el usuario escribe un comando BURN especificando explícitamente el tipo de estructura a eliminar (DATABASE o TABLE) seguido del nombre de la estructura objetivo, y procede a ejecutar el comando.

El sistema verifica la sintaxis del comando asegurándose que contenga exactamente tres tokens, y en caso de tratarse de una tabla, confirma que exista una base de datos seleccionada actualmente ya que las tablas sólo pueden eliminarse dentro del contexto de una base activa.

Posteriormente valida que la estructura especificada efectivamente existe en el sistema antes de proceder, evitando errores de eliminación de elementos inexistentes.

La solicitud de eliminación se envía al servidor que verifica los permisos del usuario y procede a eliminar tanto la definición lógica como los archivos físicos asociados del sistema de archivos.

Una vez completada la eliminación física, el sistema confirma la operación exitosa mediante mensaje al usuario, notifica a todos los usuarios conectados sobre este cambio estructural para mantener consistencia entre sesiones, actualiza la lista de estructuras disponibles en todas las interfaces conectadas, y finalmente ejecuta automáticamente RF-11 (Actualizar Cliente) para refrescar las vistas del árbol de esquemas en todos los clientes.

Este caso de uso depende de RF-01 (Iniciar Sesión) y RF-02 (Ejecutar Consultas), requiere que las estructuras hayan sido previamente creadas mediante RF-03, y al finalizar invoca RF-11 para mantener consistencia en las vistas de todos los clientes conectados.

1.3.1.6 - Insertar Registros: Este caso de uso permite poblar las tablas con información mediante el comando FILE, agregando datos a tablas existentes en el sistema.

El usuario escribe un comando FILE especificando la tabla destino, las columnas que serán afectadas, y los valores correspondientes a insertar, seguido de la ejecución del script.

El sistema inicia un proceso de validación exhaustivo: primero valida la sintaxis general del comando, luego verifica que exista una base de datos activa en la sesión actual (ya que las operaciones sobre tablas requieren un contexto de base de datos), confirma que la tabla especificada existe físicamente en el sistema de archivos, y finalmente valida que los tipos de datos de los valores proporcionados coincidan exactamente con las definiciones de las columnas de la tabla (por ejemplo, que no se intente insertar texto en una columna de tipo INT).

Una vez superadas todas las validaciones, los datos se envían al servidor para inserción, donde el servidor agrega los registros al archivo CSV asociado a la tabla, actualizando el almacenamiento físico.

El sistema confirma la inserción exitosa y proporciona retroalimentación informando la cantidad exacta de registros que fueron insertados en la operación.

Finalmente, se ejecuta RF-11 (Actualizar Cliente) para notificar los cambios a todos los usuarios conectados y mantener sincronizadas las vistas del sistema.

Este caso de uso requiere RF-01 (Iniciar Sesión) como precondición, utiliza RF-02 (Ejecutar Consultas) como mecanismo, necesita que las estructuras hayan sido creadas previamente mediante RF-03 (Definir Estructura), invoca RF-11 al completarse, y los datos insertados son posteriormente recuperables mediante RF-06 (Seleccionar Registros).

1.3.1.7 - Seleccionar Registros: Este caso de uso implementa la funcionalidad de lectura de datos, permitiendo al usuario consultar y visualizar información almacenada en las tablas mediante el comando BRING.
El usuario escribe un comando BRING especificando la tabla a consultar y las columnas específicas que desea visualizar en el resultado, seguido de la ejecución del comando.

El sistema inicia un proceso de validación múltiple: verifica la sintaxis del comando asegurándose que cumpla con las reglas gramaticales de MythQL, valida la existencia de una base de datos activa en la sesión actual (requisito para operar sobre tablas), confirma que la tabla especificada existe físicamente en la base activa, y valida que todas las columnas solicitadas en el comando existan realmente en la definición estructural de la tabla para evitar solicitudes de columnas inexistentes.

Una vez superadas estas validaciones, la consulta se envía al servidor que procesa la solicitud, recupera los datos del archivo CSV correspondiente a la tabla, filtra las columnas según lo solicitado, y retorna la información.

El sistema recibe los datos y los formatea en una estructura tabular legible, presentando finalmente los resultados en la consola del cliente con formato apropiado para facilitar su interpretación.

Este caso de uso depende de RF-01 (Iniciar Sesión) y RF-02 (Ejecutar Consultas) como precondiciones, requiere que las estructuras hayan sido creadas mediante RF-03 (Definir Estructura), y opera específicamente sobre datos que fueron previamente insertados mediante RF-05 (Insertar Registros).

1.3.1.8 - Mostrar Tablas o Base de Datos: Este caso de uso proporciona funcionalidad de navegación y exploración del sistema, permitiendo al usuario listar todas las bases de datos del sistema o todas las tablas de la base activa mediante el comando MANIFEST.

El usuario ejecuta el comando MANIFEST seguido de la palabra clave DATABASES (para listar todas las bases de datos disponibles en el sistema) o TABLES (para listar todas las tablas de la base de datos actualmente seleccionada).

El sistema verifica la sintaxis del comando asegurándose que contenga exactamente dos tokens y que el segundo token sea válido (DATABASES o TABLES), identifica qué tipo de listado se solicita según la palabra clave utilizada, y en caso de solicitar TABLES valida que exista una base de datos activa en la sesión.

El sistema obtiene la lista correspondiente del servidor: para DATABASES recupera todas las bases de datos disponibles en el sistema, mientras que para TABLES recupera todas las tablas definidas dentro de la base de datos activa.

La información se formatea para presentación clara y estructurada, y finalmente la lista se muestra en la consola del cliente permitiendo al usuario conocer qué estructuras están disponibles para trabajar.

Este caso de uso depende de RF-01 (Iniciar Sesión) y RF-02 (Ejecutar Consultas) como precondiciones, muestra los resultados de las operaciones realizadas mediante RF-03 (Definir Estructura), RF-04 (Eliminar Estructura) y RF-05 (Insertar Registros), y proporciona información contextual útil para RF-06, RF-09 y otros casos de uso que requieren conocer las estructuras disponibles antes de operar sobre ellas.

1.3.1.9 - Describir Tabla: Este caso de uso permite al usuario obtener información detallada sobre la estructura de una tabla específica mediante el comando DEPICT, revelando el esquema completo incluyendo nombres de columnas, tipos de datos y restricciones.

El usuario ejecuta el comando DEPICT seguido del nombre de la tabla que desea inspeccionar.

El sistema verifica la sintaxis del comando asegurándose que contenga exactamente dos tokens, valida que el nombre de tabla no contenga caracteres inválidos que puedan comprometer la seguridad o funcionamiento del sistema, y confirma que exista una base de datos activa en la sesión actual ya que las tablas solo pueden consultarse dentro del contexto de una base.

Posteriormente verifica que la tabla especificada existe efectivamente en la base de datos activa consultando los metadatos almacenados.

El sistema solicita al servidor la definición completa de la tabla, que recupera la información del archivo de metadatos de la base de datos donde se almacenan las definiciones estructurales.

La información de columnas (nombres, tipos de datos, restricciones) se formatea adecuadamente para presentación legible, organizando los datos en una estructura clara que muestre cada columna con sus atributos.

Finalmente, la descripción completa se muestra en la consola del cliente, proporcionando al usuario una vista detallada de la estructura interna de la tabla.

Este caso de uso requiere RF-01 (Iniciar Sesión) y RF-02 (Ejecutar Consultas) como precondiciones, opera sobre tablas creadas mediante RF-03 (Definir Estructura), y proporciona metadatos cruciales para ejecutar correctamente RF-05 (Insertar Registros) y RF-06 (Seleccionar Registros) ya que informa sobre las columnas disponibles y sus tipos de datos permitiendo al usuario construir comandos válidos.

1.3.1.10 - Actualizar Cliente: Este caso de uso implementa el mecanismo de sincronización entre clientes y servidor, permitiendo actualizar la vista del árbol de esquemas en el cliente con el estado actual del servidor. El caso puede ser invocado de dos formas: manualmente cuando el usuario hace clic en el botón "Refresh Schemas" de la interfaz, o automáticamente cuando el sistema lo invoca desde RF-03 (Definir Estructura), RF-04 (Eliminar Estructura) o RF-05 (Insertar Registros) tras modificar estructuras.

El sistema envía una solicitud GET_SCHEMAS al servidor incluyendo el token de sesión del usuario para autenticación.

El servidor valida el token asegurándose que la sesión sea válida y no haya expirado, y procede a recorrer exhaustivamente todas las bases de datos existentes y sus respectivas tablas, construyendo una estructura jerárquica completa del sistema.

El servidor retorna esta estructura en formato compacto optimizado para transmisión por red.

El cliente recibe la respuesta y la parsea, extrayendo la información jerárquica de bases de datos y tablas.

Con esta información actualizada, el cliente reconstruye completamente el árbol de esquemas en la interfaz, reemplazando la vista anterior con la nueva estructura.

El sistema expande automáticamente todos los nodos del árbol para facilitar la visualización inmediata de todas las estructuras disponibles, y finalmente muestra un mensaje "Esquemas actualizados" en la consola confirmando al usuario que la vista ha sido refrescada exitosamente.

Este caso de uso depende de RF-01 (Iniciar Sesión) como precondición absoluta, es invocado automáticamente por RF-03, RF-04 y RF-05 tras la ejecución exitosa de operaciones que modifican estructuras, y mantiene consistencia con los resultados visibles en RF-08 (Mostrar Tablas o Base de Datos) garantizando que todos los clientes conectados tengan una vista actualizada y sincronizada del estado del sistema.

1.3.1.11 - Guardar Consulta: Este caso de uso permite al usuario guardar consultas escritas en lenguaje MythQL para su uso posterior, almacenándolas como archivos con extensión .MQL en el sistema de archivos local.

El proceso inicia cuando el usuario hace clic en el botón "Guardar Query" dentro de la interfaz de MythQL, momento en el cual el sistema verifica que la consola de comandos contenga texto para guardar (un query no vacío).

Si existe contenido, el sistema abre un prompt del sistema operativo que permite al usuario seleccionar la ubicación del sistema de archivos donde desea almacenar el archivo y especificar el nombre que desea asignarle, ofreciendo una experiencia familiar de diálogo de guardado estándar.

El usuario navega por las carpetas, selecciona la ubicación deseada, ingresa el nombre del archivo, y toca el botón aceptar del diálogo.

El sistema procede a escribir el contenido de la consulta actual en un archivo con la extensión .MQL en la ubicación especificada, y finalmente devuelve un mensaje de éxito en la consola confirmando que el archivo ha sido guardado correctamente.

Este caso de uso requiere RF-01 (Iniciar Sesión) como precondición para acceder al sistema, y trabaja complementariamente con RF-11 (Abrir Consulta) permitiendo un flujo de trabajo donde los usuarios pueden guardar consultas complejas para reutilizarlas posteriormente sin necesidad de reescribirlas, facilitando la gestión y organización de scripts de base de datos.

1.3.1.12 - Abrir Consulta: Este caso de uso permite al usuario cargar consultas previamente guardadas con extensión .MQL desde el sistema de archivos local hacia la interfaz de MythQL para su visualización, edición y ejecución.

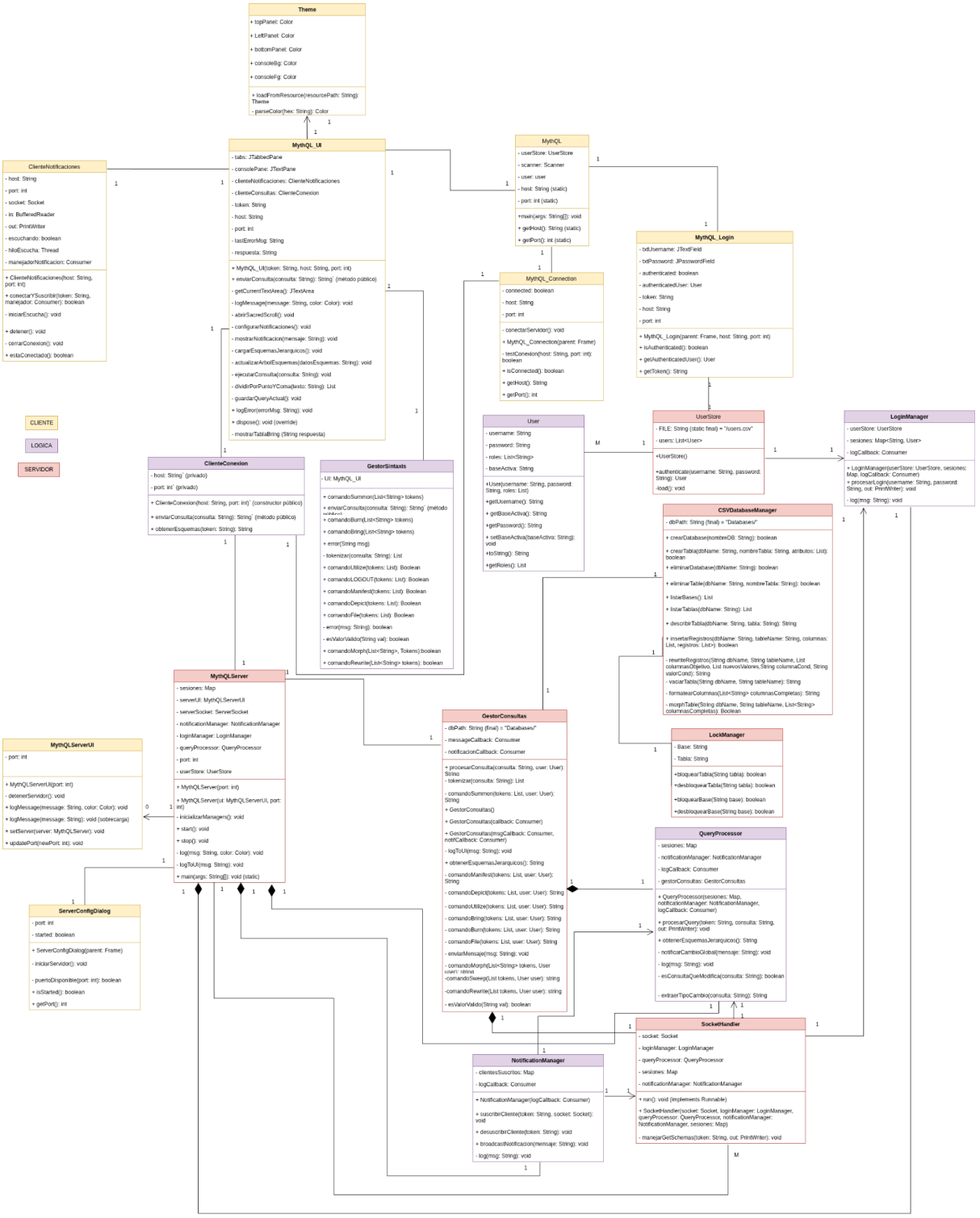
El proceso comienza cuando el usuario hace clic en el botón "Abrir Query" dentro de la interfaz de MythQL, momento en el cual el sistema despliega un prompt del sistema operativo que permite al usuario navegar por el sistema de archivos para localizar y seleccionar el archivo .MQL que desea abrir.

El usuario navega por las carpetas del sistema, selecciona el archivo de consulta deseado, y toca el botón aceptar del diálogo. El sistema valida que el archivo seleccionado tenga efectivamente la extensión .MQL para asegurar que se trata de un archivo de consulta válido de MythQL, lee el contenido del archivo desde el disco, y carga el texto de la consulta en la pestaña actual de la consola de comandos, reemplazando cualquier contenido previo que pudiera existir.

Finalmente, el sistema devuelve un mensaje de éxito en la consola informando al usuario que el archivo ha sido cargado correctamente y está listo para ser ejecutado o modificado.

Este caso de uso requiere RF-01 (Iniciar Sesión) como precondición para acceder al sistema, trabaja complementariamente con RF-10 (Guardar Consulta) permitiendo recuperar consultas guardadas previamente, y una vez cargada la consulta el usuario puede utilizar RF-02 (Ejecutar Consultas) para procesarla contra el servidor de base de datos, facilitando la reutilización de scripts complejos y el mantenimiento de bibliotecas de consultas frecuentemente utilizadas.

1.3.2 - Vista de Clases



1.3.2.1 - Estructura/Jerarquia

El sistema MYTHQL sigue una arquitectura Cliente-Servidor con separación clara de responsabilidades:

Sistema MYTHQL

```
├── Servidor (mythqlserver)
│   ├── Capa de Red y Conexiones
│   ├── Capa de Autenticación y Sesiones
│   ├── Capa de Procesamiento de Consultas
│   ├── Capa de Gestión de Base de Datos
│   └── Capa de Transacciones y Notificaciones
│
└── Cliente (MythQLPackage)
    ├── Capa de Interfaz de Usuario
    ├── Capa de Comunicación con Servidor
    ├── Capa de Validación Sintáctica
    └── Capa de Gestión de Estado
src/
├── mythqlserver/          # Servidor de base de datos
│   ├── MythQLServer.java          # Servidor principal
│   ├── SocketHandler.java         # Manejador de conexiones
│   ├── LoginManager.java          # Gestor de autenticación
│   ├── QueryProcessor.java        # Procesador de consultas
│   ├── GestorConsultas.java       # Ejecutor de comandos MYTHQL
│   ├── CSVDatabaseManager.java    # Gestor de persistencia CSV
│   ├── TransactionManager.java    # Administrador de transacciones
│   ├── NotificationManager.java   # Gestor de notificaciones
│   ├── UserStore.java             # Almacén de usuarios
│   ├── User.java                  # Modelo de usuario
│   ├── LockManager.java           # Gestor de bloqueos
│   ├── MythQLServerUI.java        # Interfaz del servidor
│   └── ServerConfigDialog.java    # Configuración del servidor
│
└── MythQLPackage/         # Cliente gráfico
    ├── MythQL.java                # Aplicación principal
    ├── MythQL_Connection.java     # Configuración de conexión
    ├── MythQL_Login.java          # Autenticación
    ├── MythQL_UI.java             # Interfaz principal
    ├── ClienteConexion.java       # Cliente para consultas
    ├── ClienteNotificaciones.java # Cliente para notificaciones
    ├── GestorSintaxis.java        # Validador sintáctico
    └── Theme.java                 # Gestor de temas
```

1.3.2.2 - Clases

1.3.2.2.1 Clases Principales del Servidor

MythQLServer - Servidor Principal

- Responsabilidad: Coordinar todos los componentes del servidor y manejar conexiones entrantes
- Atributos clave:

- `serverSocket`: Socket del servidor
- `sesiones`: Mapa de tokens de sesión → usuarios
- `userStore`: Almacén de credenciales
- Managers especializados (Login, Query, Transaction, Notification)

SocketHandler - Manejador de Conexiones

- Responsabilidad: Procesar mensajes de clientes conectados
- Métodos principales: `run()` - Bucle principal de atención al cliente

QueryProcessor - Procesador de Consultas

- Responsabilidad: Validar permisos y dirigir consultas al gestor apropiado
- Funcionalidad: Control de acceso, manejo de transacciones, logging

GestorConsultas - Ejecutor de Comandos

- Responsabilidad: Interpretar y ejecutar comandos MYTHQL
- Métodos clave: `procesarConsulta()` - Dispatcher de comandos

TransactionManager - Gestor de Transacciones

- Responsabilidad: Coordinar START/SEAL/UNDO de transacciones
- Característica: Sistema de versionado con backups incrementales

1.3.2.2.2. Clases Principales del Cliente

MythQL - Aplicación Principal

- Responsabilidad: Coordinar el flujo de la aplicación cliente
- Secuencia: Conexión → Login → Interfaz principal

MythQL_UI - Interfaz Gráfica Principal

- Responsabilidad: Proporcionar interfaz completa para interactuar con MYTHQL
- Componentes:
  - Sistema de pestañas dinámicas
  - Árbol de esquemas jerárquico
  - Consola de resultados
  - Panel de configuración

ClienteConexion - Cliente de Consultas

- Responsabilidad: Comunicación síncrona con el servidor para consultas
- Métodos: `enviarConsultaConToken()`, `obtenerEsquemas()`

ClienteNotificaciones - Cliente de Notificaciones

- *Responsabilidad: Conexión asíncrona para recibir notificaciones en tiempo real*
- *Característica: Ejecuta en hilo separado para no bloquear la UI*

GestorSintaxis - Validador Sintáctico

- *Responsabilidad: Validar localmente la sintaxis de comandos MYTHQL*
- *Método clave:* `enviarConsulta()` *- Valida antes de enviar al servidor*

1.3.2.3 - Relaciones entre clases y llamadas

1.3.2.3.1. Diagrama de Relaciones Principales

Cliente (MythQL_UI)
├── Consulta → ClienteConexion → Servidor (SocketHandler)
│ │
├── Notificaciones ← ClienteNotificaciones ← NotificationManager
│
└── Validación → GestorSintaxis

Servidor (SocketHandler)
├── Autenticación → LoginManager → UserStore
│
├── Consulta → QueryProcessor → GestorConsultas → CSVDatabaseManager
│ │
└── Transacciones → TransactionManager

1.3.2.3.2. Flujos de Comunicación

Flujo de Autenticación:

MythQL_Connection → ClienteConexion → SocketHandler → LoginManager → UserStore

Flujo de Ejecución de Consulta:

MythQL_UI → GestorSintaxis → ClienteConexion → SocketHandler → QueryProcessor → GestorConsultas → CSVDatabaseManager

Flujo de Notificaciones:

NotificationManager → ClienteNotificaciones → MythQL_UI (callback)

1.3.2.3.3. Relaciones de Dependencia

- *MythQLServer depende de:* `SocketHandler`, `LoginManager`, `QueryProcessor`, `NotificationManager`
- *QueryProcessor depende de:* `GestorConsultas`, `TransactionManager`, `NotificationManager`
- *MythQL_UI depende de:* `ClienteConexion`, `ClienteNotificaciones`, `GestorSintaxis`, `Theme`

1.3.2.4 - Procesos y Funciones

1.3.2.4.1 Procesos Principales del Servidor

1. Proceso de Inicio del Servidor

```
// MythQLServer.start()
serverSocket = new ServerSocket(port);
while (true) {
    Socket socket = serverSocket.accept();
    new Thread(new SocketHandler(socket, ...)).start();
}
```

2. Proceso de Autenticación

```
// LoginManager.procesarLogin()
User user = userStore.authenticate(username, password);
String token = UUID.randomUUID().toString();
sesiones.put(token, user);
```

3. Proceso de Ejecución de Consulta

```
// QueryProcessor.procesarQuery()
```

```
if (!tienePermisoParaConsulta(user, consulta)) return;
if (esComandoTransaccion(consulta)) {
    manejarTransaccion(token, consulta, out);
} else {
    String resultado = gestorConsultas.procesarConsulta(consulta, user);
    out.println("RESULT " + resultado);
}
```

4. Proceso de Transacción

```
// TransactionManager.startTransaction()
UserTransactionState state = new UserTransactionState(...);
userTransactions.put(token, state);
Files.createDirectories(Paths.get(backupPath));
```

1.3.2.4.2. Procesos Principales del Cliente

1. Proceso de Conexión y Login

```
// MythQL.main()
MythQL_Connection → MythQL_Login → MythQL_UI
```

2. Proceso de Ejecución de Consulta en Cliente

```
// MythQL_UI.ejecutarConsulta()
if (GestorSintaxis.valida(consulta)) {
    String respuesta = ClienteConexion.enviarConsultaConToken(token, consulta);
    mostrarResultado(respuesta);
}
```

3. Proceso de Resaltado de Sintaxis

```
// MythQL_UI.resaltarKeywordsCompleto()
Timer(300ms) → analizar texto → aplicar estilos por categoría
```

1.3.2.4.3. Funciones Clave del Sistema

GestorConsultas - Dispatcher de Comandos

```
public String procesarConsulta(String consulta, User user) {
    switch (comando.toUpperCase()) {
        case "SUMMON": return comandoSummon(tokens, user);
        case "BRING": return comandoBring(tokens, user);
        case "FILE": return comandoFile(tokens, user);
        // ... 15+ comandos soportados
    }
}
```

CSVDatabaseManager - Operaciones CRUD

```
public boolean crearDatabase(String nombreDB)
public boolean crearTabla(String dbName, String nombreTabla, List<String>
atributos)
public boolean insertarRegistros(String dbName, String tableName, ...)
public List<String> listarBases()
```

1.3.2.5 - Procesos Background

1.3.2.5.1. Procesos en Segundo Plano del Servidor

1. Manejo de Conexiones Concurrentes

```
// MythQLServer - Por cada cliente nuevo
new Thread(new SocketHandler(socket, ...)).start();
```

- *Propósito: Atender múltiples clientes simultáneamente*
- *Característica: Modelo de hilos por conexión*

2. Sistema de Notificaciones en Tiempo Real

```
// NotificationManager.broadcastNotificacion()
for (Map.Entry<String, Socket> entry : clientesSuscritos.entrySet()) {
    if (!socket.isClosed()) {
        out.println("NOTIFICATION " + mensaje);
    }
}
```

- *Propósito: Notificar cambios a todos los clientes suscritos*
- *Ejecución: Se activa tras modificaciones en la base de datos*

3. Limpieza de Recursos de Transacciones

```
// TransactionManager.cleanupUserData()
userTransactions.remove(token);
deleteDirectory(userDirFile);
```

- *Propósito: Liberar recursos cuando usuario cierra sesión*
- *Ejecución: Al recibir comando LOGOUT*

1.3.2.5.2. Procesos en Segundo Plano del Cliente

1. Recepción de Notificaciones Asíncronas

```
// ClienteNotificaciones - Hilo dedicado
new Thread(() -> {
    while (escuchando && (linea = in.readLine()) != null) {
        if (linea.startsWith("NOTIFICATION ")) {
            manejadorNotificacion.accept(notificacion);
        }
    }
}).start();
```

- *Propósito: Escuchar notificaciones sin bloquear la interfaz*
- *Característica: Ejecución continua en background*

2. Resaltado de Sintaxis con Retardo

```
// MythQL_UI - Timer para resaltado
highlightTimer = new Timer(300, evt -> {
    resaltarKeywordsCompleto(queryPane);
});
```

- *Propósito: Mejorar rendimiento al escribir*
- *Característica: Se ejecuta después de 300ms de inactividad*

3. Actualización Automática de Esquemas

```
// MythQL_UI - Al recibir notificación de cambio
SwingUtilities.invokeLater(() -> {
    cargarEsquemasJerarquicos();
});
```

- *Propósito: Mantener árbol de esquemas actualizado*
- *Activación: Por notificaciones del servidor*

1.3.2.5.3. Procesos de Mantenimiento

1. Backup Automático en Transacciones

```
// TransactionManager.backupBeforeModification()
if (transactionManager != null && transactionManager.isTransactionActive(token)) {
    transactionManager.backupBeforeModification(token, database, table);
}
```

- *Propósito: Crear puntos de restauración antes de modificaciones*
- *Ejecución: Automático al ejecutar comandos que modifican datos*

2. Gestión de Bloqueos Concurrentes

```
// LockManager.bloquearTabla()
ReentrantLock lock = tablaLocks.computeIfAbsent(nombreTabla, k -> new
ReentrantLock());
lock.lock();
```

- *Propósito: Prevenir condiciones de carrera en acceso a tablas*
- *Ejecución: Automático en operaciones de base de datos*

2. Instalación del Programa

*Para proceder con la instalación de **MythQL**, el usuario deberá acceder al sitio web oficial del proyecto, disponible en https://mythql.org. Desde allí, podrá encontrar todos los recursos necesarios para iniciar la instalación, incluyendo los archivos y documentación complementaria.*

*Una vez dentro del sitio, será necesario descargar la **máquina virtual (Virtual Machine)** que contiene el entorno preconfigurado de MythQL. Esto se realiza seleccionando la opción **"Download VM"**, ubicada en la sección de descargas principales. La descarga incluirá una imagen lista para ser ejecutada en cualquier software compatible con máquinas virtuales.*

*Tras completar la descarga, el siguiente paso consiste en **inyectar o importar la máquina virtual** en un programa que permita manipular este tipo de imágenes. Uno de los entornos más recomendados para esta tarea es **Oracle VirtualBox**, aunque*

también pueden utilizarse otras herramientas equivalentes que soporten archivos de imagen de máquinas virtuales.

*Una vez importada la imagen y ejecutada la máquina virtual, el usuario accederá al entorno del sistema operativo **Ubuntu**, que ya viene preconfigurado con todo lo necesario para correr MythQL correctamente. Dentro de este entorno, el programa **MythQL** se encontrará alojado en la **carpeta de descargas (Downloads)**, desde donde podrá iniciarse para su uso normal.*

Este proceso de instalación garantiza que MythQL funcione en un entorno controlado, con todas sus dependencias correctamente configuradas, evitando así conflictos con el sistema operativo principal del usuario. De esta manera, se asegura una instalación rápida, segura y lista para usar sin necesidad de configuraciones adicionales complejas.

3. Requerimientos

*En este apartado se especifican los **requisitos mínimos y obligatorios** necesarios para poder ejecutar y manipular **MythQL**, ya sea desde una máquina virtual (Virtual Machine) o de forma local descargando el archivo .ZIP disponible en el repositorio de GitHub del proyecto.*

Se diferencia la instalación en una VM de la ejecución directamente en el equipo del usuario para clarificar las necesidades de hardware y software en cada caso.

3.1 Requerimientos de hardware

***Memoria RAM:** 4 GB como mínimo. Recomendado: 8 GB para un rendimiento más fluido si se ejecutan otras aplicaciones simultáneamente o si la VM consume recursos adicionales.*
***Espacio en disco:** Si se utiliza **una máquina virtual**: al menos **10 GB** libres en el disco para alojar la imagen de la VM y los ficheros temporales. Si se usa la **instalación local desde el zip**: el archivo comprimido ocupa **2.53 MB**, por lo que el requisito de espacio es mínimo; aun así, se recomienda disponer de espacio adicional para bases de datos, backups y archivos generados por la aplicación.*

3.2 Requerimientos de software

***Sistemas operativos compatibles:** Ubuntu (distribuciones modernas soportadas) y Windows 10 y 11. Nota: es posible que versiones anteriores de Windows u otras distribuciones de Linux funcionen correctamente, pero el soporte y las pruebas se garantizan sobre los sistemas indicados.*

***Java:** Entorno de ejecución Java (JRE) instalado. Se recomienda usar una versión compatible con el desarrollo de MythQL (por ejemplo, Java 8 o superior—ver documentación del repositorio para la versión exacta requerida).*

NetBeans: *IDE NetBeans instalado si el usuario desea compilar o modificar el código fuente desde el entorno de desarrollo (opcional para usuarios que solo quieran ejecutar la aplicación ya compilada).*

3.3 Consideraciones adicionales

Los requisitos presentados son ***intencionalmente simples*** *y están destinados a cubrir los entornos de uso más comunes sin necesidad de equipos especializados.*

Antes de la instalación, se sugiere revisar la ***documentación del repositorio*** *en GitHub para confirmar versiones específicas de Java o dependencias adicionales que puedan incorporarse en futuras actualizaciones.*

Si se opta por ejecutar MythQL en una VM, verificar que la máquina anfitriona tenga recursos suficientes para dedicar a la VM (memoria y CPU) sin afectar el rendimiento general del sistema.

4. Configuración del Programa (Setup)

Antes de poder utilizar el programa como fue diseñado, es necesario primero inyectar en el Netbeans para Java, el proyecto de MythQL, tanto el proyecto de Cliente como de Servidor.

Es el mismo caso para la Virtual Machine. Al tener este paso ya listo, lo que se debe hacer primero es, ejecutar únicamente en el proyecto de servidor, el .JAVA llamado "MythQLServer". Lo que hace este archivo .JAVA, es habilitar un puerto para poder establecer una conexión en tu computadora para empezar a usar MythQL, una vez ya tengas establecido el puerto (A elección), deberás dejar activa la UI de fondo, ya que este actúa como proceso background ante todas las gestiones dentro del gestor de base de datos de MythQL.

Una vez con el servidor prendido, se deberá ejecutar el .JAVA en el proyecto de Cliente llamado "MythQL". Abriendo así la interfaz principal para iniciar sesión y seleccionar el puerto. Una vez seleccionado el puerto, se inicia sesión. Para caso experimental, se deberá loguearse con "admin" y su contraseña "1234". En caso contrario de no haber prendido el servidor previamente, se mostrará un error para hacer aquello.

Haber seguido el protocolo mencionado arriba, el programa debería poder abrirse normalmente, con su consola y panel de opciones.

5. Funcionalidad

En este apartado se explicará en detalle la funcionalidad que el sistema ***MythQL*** *ofrece al usuario al momento de interactuar y manipularlo. Se abordarán las capacidades, características y herramientas disponibles que permiten al usuario*

aprovechar de manera completa el potencial del sistema, garantizando una experiencia eficiente, intuitiva y alineada con los objetivos planteados en el proyecto.

***MythQL** tiene como propósito principal cumplir de forma integral con todas las tareas y requisitos establecidos en la consigna general proporcionada por nuestro cliente. Su desarrollo ha sido orientado hacia la creación de un entorno funcional capaz de ejecutar operaciones propias de un lenguaje de gestión de bases de datos, similar en comportamiento y estructura a sistemas ampliamente reconocidos como **MySQL**. Esto permite que MythQL no solo se ajuste a los estándares esperados, sino que también mantenga una lógica familiar para los usuarios con experiencia previa en dichos entornos.*

*Además, **MythQL** ofrece una interfaz práctica, sólida y accesible, que facilita la ejecución de acciones fundamentales como la **creación de tablas y bases de datos**, la **gestión y manipulación de registros**, así como diversas operaciones relacionadas con la administración de información. Su diseño busca equilibrar la **simplicidad de uso** con la **potencia funcional**, de modo que tanto usuarios principiantes como avanzados puedan emplear el sistema sin dificultad, maximizando la productividad y minimizando los posibles errores de operación.*

6. Bibliografía y Referencias

Consignas del Cliente	***Trabajo Grupal - Pro…***
DDL MythQL	***MythQL DDL***
DCU MythQL	***DCU_MythQL***
Página MythQL	***https://mythql.org/***
UML	***https://www.uml-diagrams.org/***
Guia De Estudio	***ADM Y GESTION DE…***
Guia De Estudio PT. 2	***ADM Y GESTION DE…***

MythQL Dev Team
Longaniza 4 Division