UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Fundamentos de Bases de Datos

Proyecto Final: Sistema de Tickets TI

Alumnos:

- Paredes Zamudio Luis Daniel 318159926
- Rivera Morales David 320176876
- Tapia Hernandez Carlos Alberto 315122699

Profesor: Javier León Cotonieto

21 de Noviembre de 2024

Índice

1. Descripción del Problema

En el día a día, en una oficina de una empresa promedio, el departamento de TI realiza para tareas para poder solucionar las problemáticas que conlleva tener equipo de cómputo, impresoras, servidores, etc, interconectadas entre si. Estas problemáticas son reportadas por usuarios de distintos rangos, y cada problemática tiene una prioridad distinta según el nivel de gravedad del incidente.

Comúnmente, estas incidencias son reportadas mediante *"tickets"*, en los cuales se describe la fecha, gravedad y explicación del incidente, la persona encargada de resolver la situación, la fecha de resolución del mismo, etc. Estos tickets no solo sirven para poder llevar un control de lo que se está realizando en el día a día del departamento, si no también para llevar un control histórico sobre que acciones se tomaron ante alguna incidencia y, de repetirse, conocer como se resolvió y/o que acciones ya se realizaron para la misma.

El proyecto consistirá en la creación de un esquema simplificado, junto a su base de datos, para un sistema de tickets para la siguiente situación.

Situación a Resolver

La empresa *Equilibra S.A de C.V* ofrece sus servicios de consultoría química, manejo de desechos químicos y de verificaciones de cumplimiento de las N.O.M. Al tener aproximadamente 30 empleados, es una PYME, por lo que no cuenta con un departamento de TI propio. Esto lleva a que *Equilibra S.A de C.V* rente servicios de este tipo a terceros. En particular, renta un servicio de *tickets* de Soporte Técnico para el mantenimiento y/o reparación del equipo de cómputo de la empresa.

La empresa *Sudo Services* es la empresa que provee servicios de TI a *Equilibra S.A de C.V*. Para cumplir con las necesidades de la empresa, el sistema de tickets de *Sudo Services* cuenta con una base de datos en donde se almacena información sobre la empresa, como *el equipo de cómputo de la empresa, el usuario que usa el equipo, y el área al que pertenece*. Se almacena también la información sobre *el ticket creado por el usuario, el técnico encargado de resolver el ticket y la categoría del mismo*.

El ticket es elemento principal del sistema, por que este posee, de cuenta propia, *ID, título, descripción, status, prioridad y fechas de creación y de cierre.* El ticket toma de terceros información como *ID del equipo a arreglar, ID del técnico asignado, categoría y ID del usuario que creó el ticket.*

2. Descripción, Requisitos y Requerimientos del Sistema

2.1. Entidad 'Área'

- Cada área debe de tener un ID único que lo identifique frente a otras. Se manejará como un valor entero (`INT`) que se será la llave primaria de la entidad.
- Cada área debe de tener un nombre único y no nulo. Esto se manejara como un `VARCHAR2(50)`

2.2. Entidad 'Técnico'

- Cada técnico debe de tener un ID único que lo identifique frente a otros. Se manejará como un valor entero (`INT`) que se será la llave primaria de la entidad.
- Los nombres y apellidos del técnico se guardarán en un `VARCHAR2(50)` c/u. Naturalmente, al ser nombres, deben ser valores no nulos pero pueden ser no únicos.
- Cada técnico debe de tener asignado un nivel de los siguientes: *Junior, Medio, Senior*. Este nivel se almacenará como un `VARCHAR2(20)`
- Cada técnico estará asignado a un área de la empresa en particular, por lo que es necesario que este valor sea una llave foránea que haga referencia al ID del Área y que sea del mismo tipo `INT`

2.3. Entidad 'UsuarioEmpleado'

- Cada usuario debe de tener un ID único que lo identifique frente a otros. Se manejará como un valor entero (`INT`) que se será la llave primaria de la entidad.
- Los nombres y apellidos del usuario se guardarán en un `VARCHAR2(50)` c/u. Naturalmente, al ser nombres, deben ser valores no nulos pero pueden ser no únicos.
- El rol del usuario en la empresa sera almacenado como un `VARCHAR2(50)` y debe ser no nulo. Esto por que todo empleado debe de tener asignado un rol.
- Cada usuario estará asignado a un área de la empresa en particular, por lo que es necesario que este valor sea una llave foránea que haga referencia al ID del Área y que sea del mismo tipo `INT`

2.4. Entidad 'EquipoComputo'

- Cada equipo de computo (laptop, PC, etc) debe de tener un ID único que lo identifique frente a otros. Se manejará como un valor entero (`INT`) que se será la llave primaria de la entidad.

- Cada equipo posee una marca, un numero de modelo, un numero de serial, una cantidad de memoria especifica y modelo de procesador. Notemos que aunque es necesario que ninguno de estos atributos sea nulo, el numero de serial es el único que se necesita que sea un valor único. Estos valores se guardarán en un `VARCHAR2(50)` c/u y, en particular, la cantidad de memoria se almacenará en un en un `VARCHAR2(20)`.

- Cada equipo estará asignado a un empleado en particular, por lo que es necesario que este valor sea una llave foránea que haga referencia al ID del empleado y que sea del mismo tipo `INT`

2.5. Entidad 'CategoriaTicket'

- Cada categoría debe de tener un ID único que la identifique frente a otras. Se manejará como un valor entero (`INT`) que se será la llave primaria de la entidad.

- Restringiremos de momento las categorias en las que un ticket se puede encontrar, siento estas *'Mantenimiento', 'Cambio de Equipo', 'Ajuste de Software', 'Ajuste de Hardware', 'Redes', 'Soporte General'*. Este valor se almacenará en un `VARCHAR2(20)`.

2.6. Entidad 'Ticket'

- Cada ticket debe de tener un ID único que lo identifique frente a otros. Se manejará como un valor entero (`INT`) que se será la llave primaria de la entidad.

- Cada ticket tendrá un titulo obligatorio no nulo de la situación a resolver. Se almacenará en un campo `VARCHAR(100)`.

- La descripción a detalle del ticket se almacenará en un campo `TEXT` que no debe ser nulo.

- El ticket solo se puede encontrar en uno de tres estados a la vez: *'Abierto', 'En Proceso', 'Cerrado'*. Este estatus se almacenará como un `VARCHAR2(20)`

- El ticket solo se puede encontrar en una de tres prioridades a la vez: *'Baja', 'Media', 'Alta'*. Esta prioridad se almacenará como un `VARCHAR2(20)`.

- El ticket contará con una fecha de creación y una fecha de cierre. Mientras que la fecha de creación se considerara como la fecha actual (por lo que no será nula), la fecha de cierre se considerará nula para ser actualizada a posteriori. Notemos que es importante que la fecha de cierre no puede ser una fecha menor a la fecha de creación. Ambas fechas serán almacenadas en un tipo `DATETIME`.

- El ticket tendrá referencia a 3 IDs de otras entidades: *EquipoComputo, Usuario* y *Categoría*, por lo que estas referencias serán del mismo tipo que las originales, `INT` no nulas.

2.7. Entidad 'Asignación'

La entidad funge como un intermedio entre Ticket y Técnico para conocer que técnico está asignado a que ticket.

- Cada asignación debe de tener una ID única que la identifique frente a otras. Se manejará como un valor entero (`INT`) que se será la llave primaria de la entidad.
- La asignación deberá contar con una fecha de asignación, que por default será la fecha al momento de crear una tupla y que no será nula. Se almacenará en un tipo `DATETIME`.
- La asignación tendrá referencia a 2 IDs de otras entidades: *Técnico* y *Ticket*, por lo que estas referencias serán del mismo tipo que las originales, `INT` no nulas.

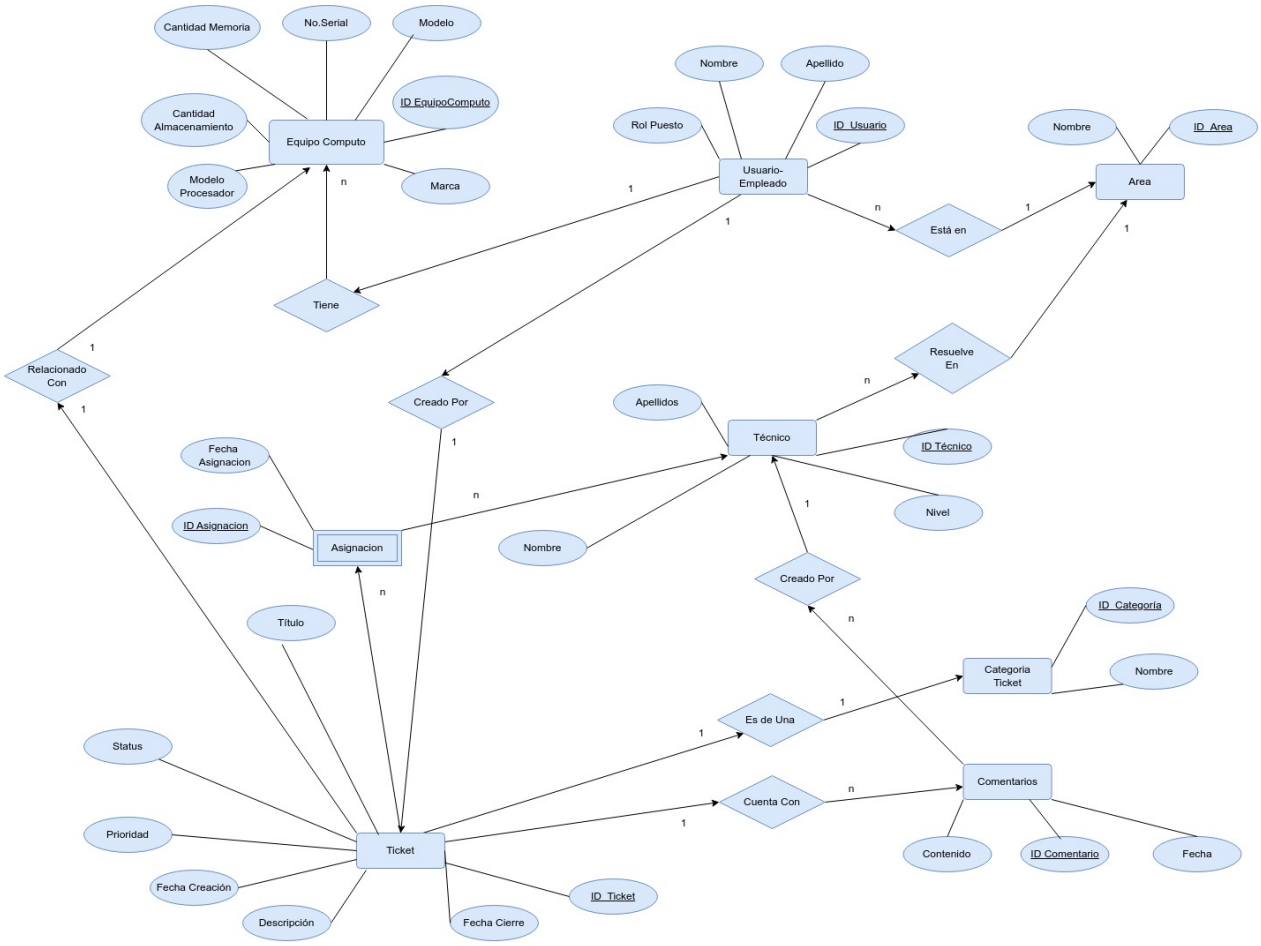
2.8. Entidad 'Comentarios'

La entidad tiene como propósito almacenar distintos comentarios sobre un mismo ticket, ya sea algún cambio al mismo, sobre como se resolvió, etc.

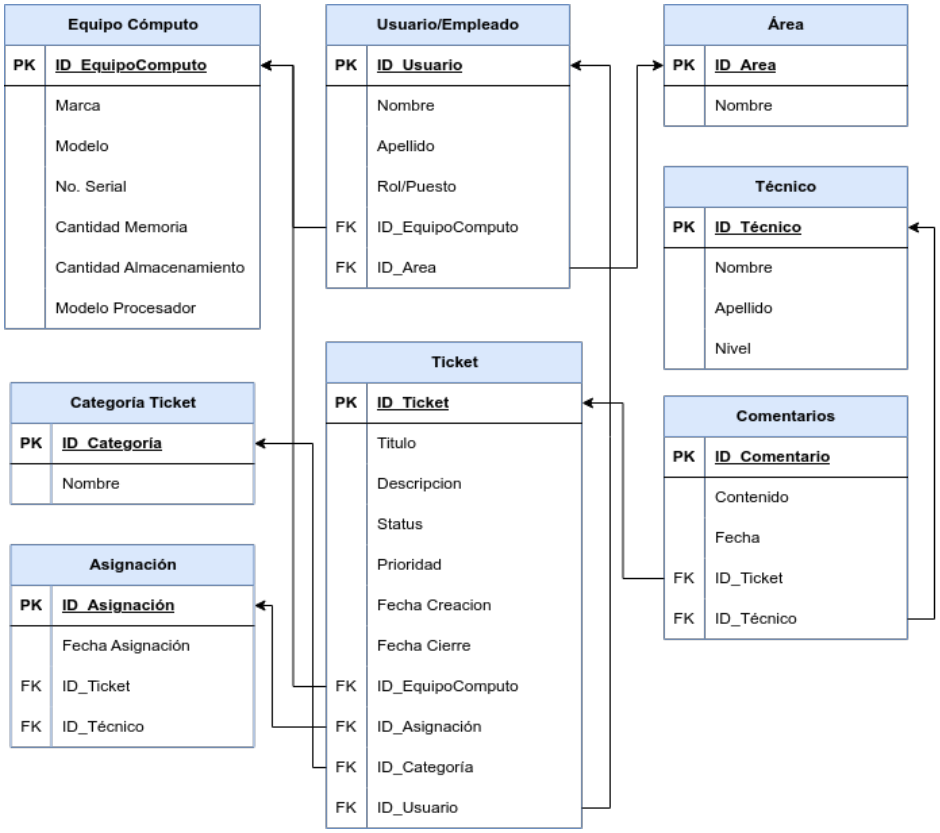
- Cada comentario debe de tener un ID único que lo identifique frente a otros. Se manejará como un valor entero (`INT`) que se será la llave primaria de la entidad.
- La descripción a detalle del comentario se almacenará en un campo `TEXT` que no debe ser nulo.
- Cada comentario deberá contar con una fecha de asignación, que por default será la fecha al momento de crearlo y que no será nula. Se almacenará en un tipo `DATETIME`.
- Cada comentario tendrá referencia a 2 IDs de otras entidades: *Técnico* y *Ticket*, por lo que estas referencias serán del mismo tipo que las originales, `INT` no nulas.

3. Modelos del Sistema

3.1. Modelo Entidad Relación


Cantidad Memoria
No.Serial
Modelo
ID EquipoComputo
Cantidad Almacenamiento
Equipo Computo
Modelo Procesador
Marca
Tiene
Relacionado Con
Nombre
Apellido
Rol Puesto
ID_Usuario
Usuario-Empleado
Está en
Nombre
ID_Area
Area
Resuelve En
Creado Por
Apellidos
Técnico
ID Técnico
Nivel
Fecha Asignacion
ID Asignacion
Asignacion
Nombre
Creado Por
Titulo
ID_Categoría
Categoria Ticket
Nombre
Es de Una
Status
Prioridad
Ticket
Cuenta Con
Comentarios
Contenido
ID Comentario
Fecha
Fecha Creación
Descripción
ID_Ticket
Fecha Cierre

3.2. Modelo Relacional



4. Nivel de Normalización del Esquema

El esquema esta normalizado en **Tercera Forma Normal (3FN)**. Consideramos que para este caso esta forma ofrece el equilibrio correcto entre *consistencia, integridad de datos y facilidad de mantenimiento* de las entidades/tablas implementadas.

Esto se justifica de la siguiente manera:

1. Se elimina redundancia innecesaria.

   - La tabla `Área` almacena únicamente la información del área, mientras que `Usuario/Empleado` usa `ID_Área` como referencia. Esto evita repetir información como el nombre del área en cada registro de usuario.

   - En `Ticket`, la categoría del ticket se gestiona con una clave foránea (`ID_Categoría`), sin duplicar los nombres de categorías en cada ticket.

   Esto reduce el espacio de almacenamiento y previene inconsistencias. Por ejemplo, si un área cambia de nombre, se actualiza en un solo lugar (`Área`), sin riesgo de que queden registros desactualizados.

2. Se garantiza integridad de datos. Gracias a la eliminación de dependencias transitivas y al uso de claves foráneas, cada tabla asegura la integridad de las relaciones. Por ejemplo:

   - En `Ticket`, las dependencias directas como `Título`, `Descripción`, y `Status` están relacionadas únicamente con `ID_Ticket`.
   - Si un ticket es asignado a un técnico (`Asignación`), este técnico está referenciado mediante `ID_Técnico`, que garantiza que exista en la tabla `Técnico`.

   Esto evita datos huérfanos o relaciones rotas,

3. Facilita el mantenimiento del esquema y reduce el riesgo de inconsistencias en los datos.

   - Las actualizaciones, inserciones o eliminaciones son más simples porque afectan menos tablas.
   - Las modificaciones no generan errores en otras partes del sistema. Por ejemplo:
     - Si el nivel de un técnico cambia, se actualiza solo en la tabla `Técnico`, sin impactar registros de tickets anteriores.

4. Existe una mejor claridad sobre las entidades y sus relaciones. Por ejemplo:

   - Los usuarios están relacionados con áreas y equipos de cómputo.
   - Los tickets están vinculados a usuarios, técnicos, equipos y categorías.
   - Cada asignación y comentario tiene un enlace directo con el ticket correspondiente.

5. Es más fácil de escalar. Se pueden agregar nuevos datos o funcionalidades sin comprometer la estructura previa del esquema. Por ejemplo:

   - Si en el futuro se desea agregar un nuevo atributo para las áreas (como ubicación), solo es necesario modificar la tabla `Área`, sin afectar las tablas `Usuario/Empleado` o `Ticket`.

5. Roles y Control de Usuarios

Como el esquema está destinado a un *departamento* TI, se entiende que existen distintos niveles de responsabilidad dentro del mismo. Eso nos lleva a que existan distintos usuarios con ciertos privilegios. Por ejemplo, un tipo de usuario puede solo tener privilegios de 'observar' para revisar tickets abiertos, cerrados y en curso asignados a él, mientras que un usuario administrador debe tener todos los permisos de escritura y lectura, o incluso debe de existir un usuario con permisos intermedios.

Para simplificar esto, hemos decidido crear 4 roles con una lista puntual de permisos para así tener un control

de acceso y modificación a la información del sistema.

5.1. AdminBD

Este rol tendría control total. Este rol permite hacer `SELECT, INSERT, DELETE, UPDATE, EXECUTE, DROP, ALTER`, etc. Puede crear nuevos técnicos y modificar información provista por la empresa. Notemos que esto no es lo mismo que el usuario `root` ya que estamos restringiendo este rol unicamente a la base de datos del Sistema de Tickets y no damos acceso a otras bases que puedan existir en la misma conexión al SMBD.

5.2. TecnicoSr

Este rol tiene control total sobre los tickets, pero no sobre la información de la empresa (por ejemplo, no puede modificar información sobre las áreas de la empresa), pero puede, por ejemplo, cambiar de equipo a un usuario. Tiene todos los permisos para trabajar unicamente sobre las tablas `Ticket, Asignacion, EquipoComputo, CategoriaTicket, Comentarios`.

5.3. TecnicoMedio

Este rol tiene un control medio sobre las tablas referentes a los tickets. *TecnicoMedio* puede asignar a otros técnicos ciertos tickets y hacer operaciones `SELECT, UPDATE, INSERT, INDEX, EXECUTE` sobre las tablas y procedimientos en `Ticket, Comentarios, Asignacion`.

5.4. TecnicoJr

Este rol es el que menos permisos tiene. Este rol esta pensando para los usuarios que solo podrán crear y editar nuevos tickets, pero no borrarlos. De esto, que el rol solo tenga permisos de `SELECT, EXECUTE` sobre la tabla `Ticket`.

Con estos roles se incluyen 4 usuarios, uno por cada rol, con el que se pueden crear conexiones al SMBD para empezar a interactuar con la base de datos:

1. `tecnicoSudo`, con permisos de TecnicoSr, con contraseña `aunNoEresRoot`.
2. `tecnicoSteveRodgers`, con permisos de TecnicoJr, con contraseña `buckyNeverDie`.
3. `tecnicoBarryAlen`, con permisos de TecnicoMedio, con contraseña `notAnotherFlashpoint`.
4. `admin`, con permisos de AdminBD, con contraseña `APbuDqoK6j5u0UyMN`.

6. Vistas Predefinidas

Se incluyen 7 vistas predefinidas para la visualización de los tickets en sus diversos estados y prioridades.

1. `TicketsActivos`: Vista para revisar la información sobre los tickets activos sin importar su prioridad.
2. `TicketsCerrados`: Vista para revisar la información sobre los tickets cerrados sin importar su prioridad.
3. `TicketsActivosAltaPrioridad`: Vista para revisar la información sobre los tickets activos con alta prioridad.
4. `TicketsActivosMediaPrioridad`: Vista para revisar la información sobre los tickets activos con prioridad media.
5. `TicketsActivosBajaPrioridad`: Vista para revisar la información sobre los tickets activos con prioridad baja.
6. `AsignacionTicketsAbiertos`: Vista para revisar las asignaciones a técnicos sobre tickets abiertos.
7. `AsignacionTicketsCerrados`: Vista para revisar las asignaciones a técnicos sobre tickets cerrados.

Para ejecutar cualquiera de las vistas, se sigue el siguiente formato: `SELECT * FROM <NombreDeLaVista>`. Por ejemplo:

```
01 |      SELECT * FROM TicketsActivos;
```

7. Funcionalidades Implementadas al Sistema

El sistema cuenta con 2 Triggers, 3 Procedimientos Almacenados y una Función implementada, c/u con una funcionalidad específica y resuelven situaciones comunes con un sistema de tickets.

7.1. Triggers

1. `NoCierreNoAsignacion`: Validación de un ticket para que no pueda cerrarse sin asignación. De esta forma, el trigger se dispara antes de actualizar un ticket, analizando si el estado nuevo del ticket es ¸cerradoz que no exista alguna asignación para el ticket, si se cumplen ambos criterios se lanza una excepción.
2. `TecnicoAsignadoComentarios`: Trigger que valida que solo el técnico asignado a un ticket pueda hacer comentarios, por lo que se dispara antes de insertar un comentario. El trigger analiza si el técnico que está intentando insertar el comentario es el técnico asignado al ticket. En caso contrario, se lanza una excepción de ejecución.

7.2. Procedimientos Almacenados

Nota Sobre los Procedimientos Almacenados

Los procedimientos almacenados implementados funcionan a base de transacciones, por lo que c/u tiene su condición y su excepción de salida en caso de no poderse ejecutar correctamente.

1. `CambiarEstadoTicket`: Procedimiento para actualizar el estado de un ticket especificado. El procedimiento toma dos parámetros, el ID del Ticket a modificar y el nuevo estado del mismo. Es importante recalcar que el estado del ticket solo puede ser 'Abierto', 'En Proceso' o 'Cerrado'. De no ingresar un estado o un ID válido, el proceso termina con erro de ejecución. Para usarlo, se puede hacer una llamada como la siguiente:

```
01 |          CALL CambiarEstadoTicket(1, 'En Proceso');
02 |
```

2. `AsignarTicket`: Procedimiento que asigna un ticket a un técnico específico. Este procedimiento valida que un técnico pertenezca al área del ticket. Toma el parámetro ID del ticket e ID del técnico. Si el ID del técnico pertenece al área, entonces entrará .en proceso", si no es así, el proceso terminará con una ejecución errónea

```
01 |          CALL AsignarTicket(1, 2);
02 |
```

3. `AgregarComentario`: Procedimiento para agregar un comentario a un ticket específico. Este procedimiento toma de parámetros el ID del ticket y luego el comentario sobre el mismo. Si el ID insertado no corresponde a un ticket existente, el proceso termina con una ejecución errónea. Se usa de la siguiente manera:

```
01 |          CALL AgregarComentario(1, 'Se ha iniciado el proceso de revision.');
02 |
```

7.3. Funciones

1. `ContarTicketsPorEstado`: Función que cuenta el número total de tickets que tienen un estado específico. Toma de parámetro el estado de los tickets a revisar y regresa un entero significando la cantidad de tickets en dicho estado. Se usa de la siguiente manera:

```
01 |          SELECT ContarTicketsPorEstado('Abierto');
02 |
```

8. Uso del Código para el Sistema

Anexado a este documento se encuentran los archivos SQL que implementan el Sistema. Para ejecutarlos se necesita contar con el SMBD DBeaver y una instancia de MariaDB o de MySQL instalada en alguna computadora.

Para poder formar el esquema de forma correcta se ejecutan los archivos ubicados en de la carpeta *Scripts* de la siguiente manera:

1. `Esquema.sql`
2. `FuncionesYProcedimientos.sql`
3. `Triggers.sql`
4. `Vistas.sql`
5. `ControlDeUsuarios.sql`
6. `Inserts.sql`

El último archivo funge como el diccionario de datos de prueba del sistema.