# Modelo de Base de Datos Sistema de Helpdesk con inventario de dispositivos y gestión de conocimiento

Entrega Proyecto Final


CODERHOUSE

Curso SQL Comisión 47360

**Profesor:** Santiago Luis Acosta Rapoani
**Tutor**: Matías Cantora
**Alumno:** Francisco Tomas Fernández Aguirre

CODERHOUSE

# Índice

CODERHOUSE

# 1. Diseño

Este modelo de base de datos, basado en MySQL. ha sido diseñado con el propósito de administrar los datos de un sistema de Helpdesk. Con esta herramienta, las organizaciones tienen la capacidad de registrar, gestionar y solucionar solicitudes de soporte, incidentes y requerimientos de manera eficiente. Además, permite el control efectivo del inventario de dispositivos tecnológicos y la gestión de conocimientos.

# 2. Objetivo

El modelo de base de datos tiene como objetivo optimizar y proporcionar una estructura para la gestión del soporte técnico y los procesos de resolución de incidentes y requerimientos. Proporciona una estructura para rastrear y administrar los tickets de manera eficiente, asegurando que cada solicitud sea atendida de manera adecuada y en función de su prioridad. Además, la integración de la base de conocimiento enriquece la experiencia al permitir el registro y acceso a soluciones previamente documentadas, acelerando la resolución de problemas y aumentando la productividad del equipo de soporte.

# 3. Tablas Principales y Relaciones:

## *3.1. Empresa*

Almacena información sobre las empresas relacionadas con el sistema. Cada empresa tiene un ID único, un nombre, área y subárea asociada. Además, se establecen relaciones con las tablas de Usuarios y Activos para asociar usuarios y activos a empresas específicas. Estas relaciones permiten organizar y filtrar la información en función de las empresas a las que pertenecen los usuarios y activos.

- Relación (1,n) con Tabla de Activos (una empresa puede estar asociado a varios activos)
- Relación (1,n) con Tabla de Usuarios (una empresa puede estar asociado a varios usuarios)

## *3.2. Usuarios*

Almacena información sobre los usuarios del sistema, incluidos sus nombres, direcciones de correo electrónico, roles cargos y áreas en las que trabajan. Cada usuario puede estar asignado a varios tickets y puede contribuir a la base de conocimiento.

- Relaciones: (1,n) con Tabla de Tickets (un usuario puede estar asociado a varios tickets)
- Relaciones: (1,n) con Tabla de Empresas (un usuario pertenece a una empresa específica, pero una empresa puede tener varios usuarios)
- Relaciones: (1,n) con Tabla de Base de Datos de Conocimiento (un usuario puede crear varios registros de conocimiento)

## *3.3. Notificaciones*

Permite el envío de notificaciones a usuarios y técnicos en función de eventos específicos, manteniendo a todos informados sobre el estado de los tickets.

- Relación (1,1) con Tabla de Usuarios (Una notificación debe estar relacionado con un único usuario)

### *3.4. Activos*

Registra detalles de los activos relacionados con los problemas o solicitudes. Activos pueden ser estaciones de trabajo como Laptop , periféricos , monitores , servidores , dispositivos de red , celulares , etc.  Los activos pueden estar asociados a múltiples tickets.

- Relaciones: (1,n) con Tabla de Tickets (un activo puede estar asociado a varios tickets)
- Relaciones: (1,n) con Tabla de Tickets (un activo puede estar asociado a varios tickets)

### *3.5. Tickets*

Captura detalles completos sobre cada ticket de soporte, incluyendo estado (abierto, solucionado, cerrado, etc.), tipo (incidente, requerimiento), categoría, prioridad y fechas importantes. Cada ticket está vinculado a un usuario asignado y un activo a su vez vinculado al, y puede tener comentarios y registros de conocimiento asociados.

- Relaciones: (1,1) con Tabla de Usuarios (un ticket debe tener un usuario asignado)
- Relaciones: (1,1) con Tabla de Activos (un ticket debe estar relacionado con un activo)
- Relaciones: (1,1) con Tabla de Categorías (un ticket debe tener una categoría)
- Relaciones: (1,1) con Tabla de Prioridades (un ticket debe tener una prioridad)
- Relaciones: (1,n) con Tabla de Comentarios (un ticket puede tener varios comentarios)

### *3.6. Comentarios*

La tabla de Comentarios se utiliza para mantener un registro de las interacciones y notas relacionadas con un ticket específico. Cada comentario puede estar asociado a un ticket, un usuario que lo realizó y una marca de tiempo que registra cuándo se hizo el comentario. Esta información es valiosa para llevar un historial detallado de las conversaciones y actividades relacionadas con los tickets de soporte.

- Relación (1,1) con Tabla de Tickets (un comentario debe estar relacionado con un ticket)
- Relación (1,1) con Tabla de Usuarios (un comentario debe tener un usuario asociado)

### *3.7. Categorías*

Proporciona la clasificación y nivel de prioridad para organizar y asignar tickets de manera eficiente. Se definieron 3 niveles de categorías. Un ejemplo seria: categoría 1 - “Hardware”, categoría 2 - “Problema con disco duro, categoría 3 – “Termino de vida útil”.

- Relaciones: (1, n) con Tabla de Tickets (una categoría puede tener varios tickets)

### *3.8. Base de Datos de Conocimiento*

Agrega la capacidad de registrar problemas y soluciones relacionados con tickets. Cada entrada de conocimiento está vinculada a uno o varios tickets, permitiendo una gestión efectiva de la información y la solución de problemas recurrentes.

- Relación (1,1) con Tabla de Tickets (un comentario debe estar relacionado con un ticket)
- Relación (1,1) con Tabla de Usuarios (un comentario debe tener un usuario asociado)

CODERHOUSE

# 4. Tablas Tipos de elementos parametrizados en tablas

Algunas tablas contienen registros que cuentan con parámetros preestablecidos, esto quiere decir que sus valores si bien son modificables por están en función de ciertos parámetros preestablecidos por el diseño de la base de datos. A continuación, se detallarán los parámetros preestablecidos de las tablas:

## *4.1. Tabla Usuarios, campo "ROL":*

- Administrador:
    - Descripción: Los administradores tienen acceso completo y privilegios especiales en el sistema.
    - Responsabilidades: Pueden gestionar usuarios, configurar permisos, acceder a todas las funciones y datos del sistema y tomar decisiones críticas para la administración de la plataforma.
    - Acceso: Acceso completo y control total sobre la plataforma.
- Técnico:
    - Descripción: Los técnicos suelen ser responsables de brindar soporte técnico o resolver problemas técnicos en el sistema.
    - Responsabilidades: Resuelven problemas, gestionan tickets, responden a consultas de usuarios y pueden tener acceso a herramientas y funciones específicas para realizar tareas técnicas.
    - Acceso: Acceso a áreas y funciones relacionadas con el soporte técnico.
- Usuario:
    - Descripción: Los usuarios son los clientes o empleados finales que utilizan la plataforma para acceder a recursos.
    - Responsabilidades: Pueden crear tickets, realizar consultas, acceder a recursos, seguir el progreso de sus solicitudes y colaborar con el equipo de soporte.
    - Acceso: Acceso limitado a las funciones necesarias para interactuar con el sistema y solicitar asistencia.

## *4.2. Tabla Notificaciones, campo "TIPO DE NOTIFICACIÓN":*

- Info:
    - Descripción: Notificación que da la bienvenida a los nuevos usuarios a la plataforma.
    - Uso: Para saludar a los nuevos usuarios, proporcionar información inicial o alguna notificación de utilidad.
- Actualización de Tickets:
    - Descripción: Notificación sobre cambios en los tickets de los usuarios.
    - Uso: Para informar a los usuarios sobre actualizaciones en sus solicitudes de soporte.
- Alerta:
    - Descripción: Notificación de eventos o información importante en la plataforma.
    - Uso: Para comunicar eventos críticos, anuncios, mantenimientos, o cualquier otra información relevante.
- Recordatorios:
    - Descripción: Notificación que recuerda a los usuarios tareas pendientes o acciones necesarias.
    - Uso: Para ayudar a los usuarios a mantenerse al tanto de sus responsabilidades en el sistema.
- Novedades o Actualizaciones de la Plataforma:
    - Descripción: Notificación sobre nuevas características, mejoras o actualizaciones en la plataforma.
    - Uso: Para informar a los usuarios sobre cambios en la plataforma que puedan afectar su experiencia.
- Eventos o Anuncios Importantes:
    - Descripción: Notificación sobre eventos significativos o anuncios de la empresa.
    - Uso: Para comunicar eventos programados, anuncios importantes u otras noticias relevantes.

### *4.3. Tabla Activos, campo "ESTADO":*

- Bueno: El activo está en un estado óptimo, funcionando correctamente y sin problemas significativos. No requiere reparaciones o mantenimiento inmediato.
- Regular: El activo aún funciona, pero puede presentar problemas menores o desgaste normal. Puede requerir mantenimiento preventivo o reparaciones menores en un futuro cercano.
- Malo: El activo presenta problemas significativos o está fuera de servicio debido a fallas importantes. Requiere reparaciones importantes o reemplazo.
- Dado de baja: El activo se ha retirado de la operación y ya no se utiliza. Puede ser desechado o almacenado de forma permanente.

### *4.4. Tabla Tickets, campos "TIPO", "PRIORIDAD" Y "ESTADO":*

- Tipo:
    - Incidente: Un tipo de ticket que se utiliza que hace referencia a un problema de funcionamiento o continuidad operacional. Esto puede ser alguna falla tanto de software o de hardware, caída de alguna plataforma o problemas de conexión, etc.
    - Requerimiento: Un tipo de ticket que se utiliza para solicitar un servicio, función o acción específica que no está relacionada con una situación. Esto puede ser modificación de contraseña, creación de cuenta, solicitud de repuesto o insumo, etc.
- Prioridad:
    - Baja: Indica que el ticket tiene baja prioridad y no es urgente. Puede manejarse en un plazo de tiempo más largo.
    - Media: Indica que el ticket tiene prioridad moderada y debe ser atendido en un plazo razonable.
    - Alta: Indica que el ticket tiene alta prioridad y debe ser atendido con prontitud.
    - Urgente: Indica que el ticket es de máxima prioridad y requiere una atención inmediata.
- Estado:
    - Abierto: El ticket ha sido creado, pero aún no se ha comenzado a trabajar en él.
    - Pendiente información: El ticket está a la espera de información adicional o detalles antes de que se pueda avanzar en su resolución. Esto es solicitud de información adicional al incidente o requerimiento necesario para su resolución.
    - Trabajo en curso: El ticket está siendo atendido y trabajando en la resolución.
    - Resuelto: El ticket ha sido resuelto, pero requiere confirmación de parte del usuario que el ticket ha sido resuelto.
    - Cerrado: El ticket se ha completado y se ha cerrado oficialmente luego que el usuario haya confirmado la solución.

### *4.5. Tabla categorías, campo "NOMBRECATEGORIA"*

- Hardware: Esta categoría se refiere a todos los componentes físicos de una computadora o dispositivo, como procesadores, memoria RAM, discos duros, tarjetas gráficas, teclados, mouse, etc. Incluye todo el equipo físico que compone una infraestructura tecnológica.
- Redes: La categoría de "Redes" se centra en todo lo relacionado con la configuración, gestión y mantenimiento de redes de comunicación. Esto incluye routers, switches, cables, protocolos de red y cualquier componente que permita la conectividad entre dispositivos.
- Software: Aquí se agrupan todos los programas, aplicaciones y sistemas operativos utilizados en una organización. Esto puede incluir desde sistemas operativos, software de productividad como suites de oficina hasta software especializado para tareas específicas.
- Plataforma: La categoría de "Plataforma" se refiere a las bases sobre las cuales se construyen aplicaciones y sistemas. En el caso de requerimientos se refiere por ejemplo a creación de usuario modificaciones de contraseña, etc. Referente a incidentes se requiere a caídas del sistema.
- Adquisiciones: Esta categoría se relaciona con la adquisición de activos y recursos, como hardware, software o cualquier otro elemento necesario para el funcionamiento de una organización. Puede incluir registros de compras, proveedores y contratos.

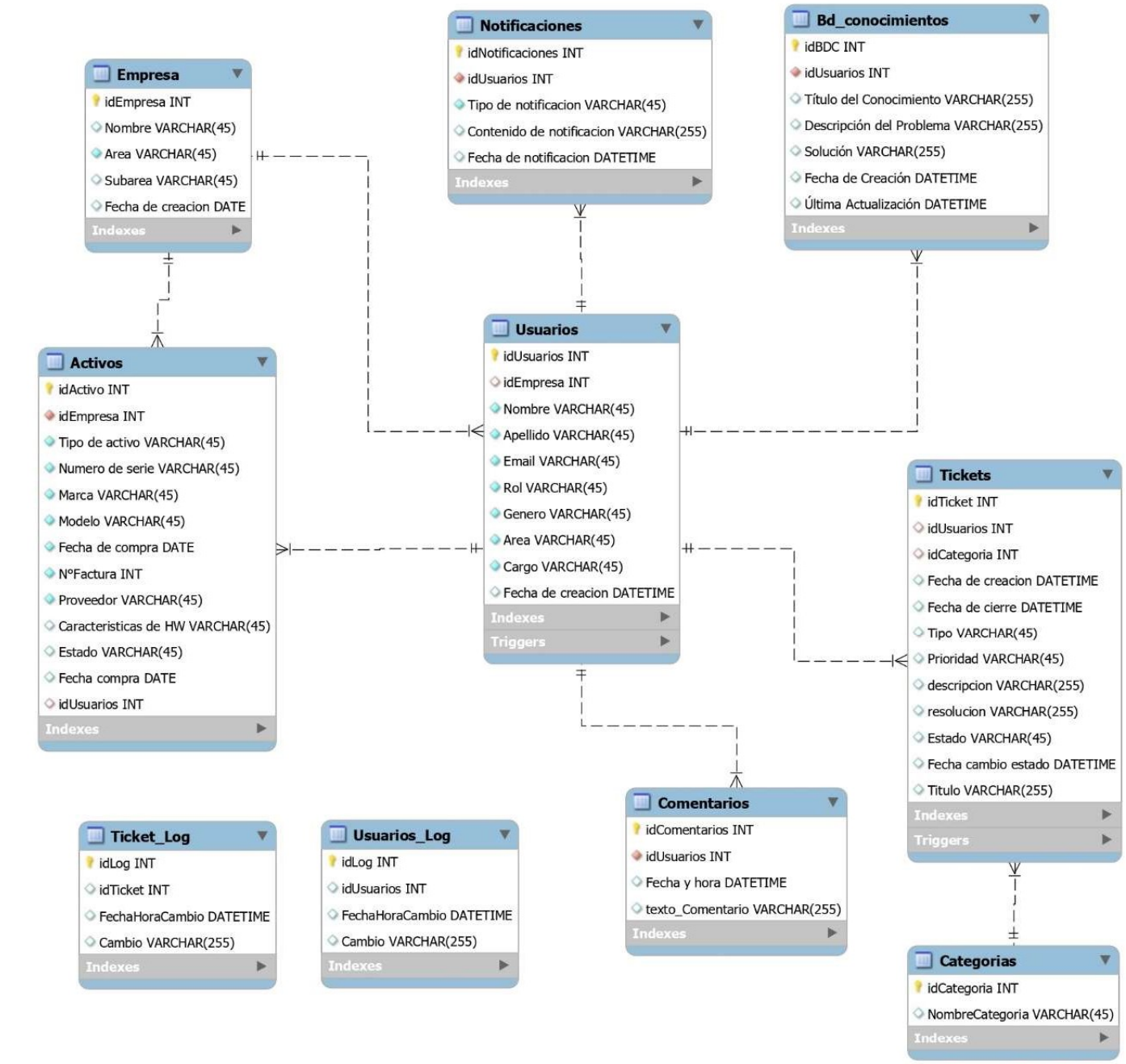
CODERHOUSE

## 5. Descripción de tablas del modelo

| TABLA | NOMBRE DEL CAMPO | CLAVE PRIMARIA | CLAVE FORANEA | TIPO DE DATO |
|---|---|---|---|---|
| **Empresa** | idEmpresa | PK | | INT |
| | Nombre | | | VARCHAR(45) |
| | Area | | | VARCHAR(45) |
| | Subarea | | | VARCHAR(45) |
| | Fecha de creacion | | | DATE |
| **Usuarios** | IdUsuarios | PK | | INT |
| | IdEmpresa | | FK | INT |
| | Nombre | | | VARCHAR(45) |
| | Apellido | | | VARCHAR(45) |
| | Email | | | VARCHAR(45) |
| | Rol | | | VARCHAR(45) |
| | Genero | | | VARCHAR(45) |
| | Area | | | VARCHAR(45) |
| | Cargo | | | VARCHAR(45) |
| | Fecha de creacion | | | DATETIME |
| **Notificaciones** | IdNotificaciones | PK | | INT |
| | idUsuarios | | FK | INT |
| | Tipo de notificacion | | | VARCHAR(45) |
| | Contenido de notificacion | | | VARCHAR(45) |
| | Fecha de notificacion | | | DATETIME |
| **Activos** | IdActivo | PK | | INT |
| | idEmpresa | | FK | INT |
| | idUsuarios | | | |
| | Tipo de activo | | | VARCHAR(45) |
| | Numero de serie | | | VARCHAR(45) |
| | Marca | | | VARCHAR(45) |
| | Modelo | | | VARCHAR(45) |
| | Fecha de compra | | | VARCHAR(45) |
| | N°Factura | | | INT |
| | Proveedor | | | VARCHAR(45) |
| | Estado | | | VARCHAR(45) |
| | Caracteristicas de HW | | | VARCHAR(45) |
| | Fecha compra | | | DATETIME |
| **Tickets** | idTicket | PK | | INT |
| | idActivo | | FK | INT |
| | idUsuarios | | FK | INT |
| | idCategoria | | FK | INT |
| | Fecha de creacion | | | DATETIME |
| | Fecha de cierre DATE | | | DATETIME |
| | Tipo | | | VARCHAR(45) |
| | Prioridad | | | INT |
| | descripcion | | | VARCHAR(255) |
| | resolucion | | | VARCHAR(255) |
| | Estado | | | VARCHAR(45) |
| | Fecha cambio estado | | | DATETIME |
| | Titulo | | | VARCHAR(45) |
| **Comentarios** | idComentarios | PK | | INT |
| | idUsuarios | | FK | INT |
| | Fecha y hora | | | DATETIME |
| | texto_Comentario | | | VARCHAR(255) |
| **Categorias** | idCategoria | PK | | INT |
| | NombreCategoria | | | VARCHAR(45) |
| **BD conocimientos** | idBDC | PK | | INT |
| | idUsuarios | | FK | INT |
| | Título del Conocimiento | | | VARCHAR(45) |
| | Descripción del Problema | | | VARCHAR(45) |
| | Solución | | | VARCHAR(45) |
| | Fecha de Creación | | | DATEDATETIME |
| | Última Actualización | | | DATETIME |
| **Usuarios_Log** | idLog | PK | | INT |
| | idUsuarios | | | INT |
| | FechaHoraCambio | | | DATETIME |
| | Cambio | | | VARCHAR(255) |
| **Ticket_Log** | idLog | PK | | INT |
| | idTicket | | | INT |
| | FechaHoraCambio | | | DATETIME |
| | Cambio | | | VARCHAR(255) |

CODERHOUSE

## 6. Modelo Entidad-Relación




vista_total_tickets_empresas

Categorias_TicketAbiertos_Urgente

vista_activos_bdconocimiento

vista_notificaciones_usuarios

vista_solucionesBDconocimiento_empresa

vista_tickets_abiertos_cocacola_y_acme

CODERHOUSE

# 7. Vistas

## *7.1. vista_tickets_abiertos_cocacola_y_acme:*

```sql
-- VISTA DE TODOS LOS TICKET  DE USUARIOS DE LA EMPRESA COCACOLA Y ACME ----------------------------------
-- Esta vista muestra todos los tickets abiertos relacionados con los usuarios de las empresas "Cocacola" y "Acme".
-- Proporciona información sobre el número de ticket, el título del ticket, el estado y el nombre de la empresa a la que están asociados
CREATE VIEW vista_tickets_abiertos_cocacola_y_acme AS
SELECT
    Tickets.idTicket AS N°Ticket,
    Tickets.Titulo AS TituloTicket,
    Tickets.Estado,
    Empresa.Nombre AS NombreEmpresa
FROM
    Tickets
INNER JOIN
    Usuarios ON Tickets.idUsuarios = Usuarios.idUsuarios
INNER JOIN
    Empresa ON Usuarios.idEmpresa = Empresa.idEmpresa
WHERE
    Empresa.Nombre = 'Cocacola' OR Empresa.Nombre = 'Acme';

select * from vista_tickets_abiertos_cocacola_y_acme;
```

## *7.2. vista_activos_bdconocimiento:*

```sql
-- VISTA DESDE QUE ACTIVOS SE CREAEON REGISTROS EN BASE DE DATOS DE CONOCIMIENTO -------------------------------
-- Esta vista relaciona los activos con los registros de la base de datos de conocimiento y muestra información relevante, como el tipo de activo,
-- el número de serie, el título del conocimiento, la descripción del problema, la fecha de creación, la última actualización y la solución asociada.
CREATE VIEW vista_activos_bdconocimiento AS
SELECT
    Activos.`Tipo de activo` AS TipodeActivo,
    Activos.`Numero de serie` AS NumerodeSerie,
    Bd_conocimientos.`Título del Conocimiento` AS TitulodelConocimiento,
    Bd_conocimientos.`Descripción del Problema` AS DescripciondelProblema,
    Bd_conocimientos.`Fecha de Creación` AS FechaCreacion,
    Bd_conocimientos.`Última Actualización` AS UltimaActualizacion,
    Bd_conocimientos.`Solución` AS Solucion
FROM
    Activos
INNER JOIN
    Bd_conocimientos ON Activos.idUsuarios = Bd_conocimientos.idUsuarios;

select * from vista_activos_bdconocimiento;
```

## *7.3. vista_notificaciones_usuarios:*

```sql
-- VISTA NOTIFICACIONES QUE HAN RECIBIDO LOS USUARIOS ----------------------------------
-- Esta vista muestra las notificaciones que los usuarios han recibido. Incluye detalles como el tipo de notificación, el contenido de la notificación,
-- la fecha de notificación y el nombre y apellido del usuario al que se dirigieron las notificaciones.
CREATE VIEW vista_notificaciones_usuarios AS
select
    Notificaciones.`Tipo de notificacion` AS TipodeNotificacion,
    Notificaciones.`Contenido de notificacion` AS ContenidoNotificacion,
    Notificaciones.`Fecha de notificacion` AS FechaNotificacion,
    Usuarios.nombre AS NomUsuario,
    Usuarios.Apellido AS Apellido
   FROM
    Usuarios
INNER JOIN
    Notificaciones ON Usuarios.idUsuarios = Notificaciones.idUsuarios;

   select * from vista_notificaciones_usuarios;
```

### 7.4. vista_categorias_TicketAbiertos_Urgente

```sql
-- CATEGORIAS DE TICKETS TIPO INCIDENTE CON ESTADO ABIERTO Y PRIORIDAD URGENTE ----------------------------
-- Esta vista se centra en los tickets de tipo "Incidente" con estado "Abierto" y prioridad "Urgente".
-- Muestra información sobre la categoría de estos tickets, su tipo y prioridad.

CREATE VIEW vista_categorias_TicketAbiertos_Urgente AS
SELECT
    Tickets.tipo AS TipoTicket,
    Tickets.Prioridad AS PrioridadTicket,
    Categorias.NombreCategoria
FROM
    Tickets
INNER JOIN
    Categorias ON Categorias.idCategoria = Tickets.idCategoria
WHERE
    Tickets.Prioridad = 'Urgente' AND Tickets.tipo = 'Incidente';

select * from Categorias_TicketAbiertos_Urgente;
```

### 7.5. vista_solucionesBDconocimiento_empresa

```sql
-- VISTA DE SOLUCIONES DE BASE DE DATOS DE CONOCIMIENTO QUE HAN CARGADO POR EMPRESA ------------------------------
-- Esta vista presenta las soluciones de la base de datos de conocimiento que han sido cargadas por empresas.
-- Incluye detalles como el título del conocimiento,
-- la descripción del problema, la solución y el nombre de la empresa que contribuyó a la carga de la solución.

CREATE VIEW vista_solucionesBDconocimiento_empresa AS
SELECT
    Bd_conocimientos.`Título del Conocimiento` AS Titulo,
    Bd_conocimientos.`Descripción del Problema` AS Descripcion,
    Bd_conocimientos.Solución AS Solucion,
    Empresa.nombre as NombreEmpresa
FROM
    Bd_conocimientos
INNER JOIN
    Usuarios ON Usuarios.idUsuarios = Bd_conocimientos.idUsuarios
INNER JOIN
    Empresa ON Empresa.idEmpresa = Usuarios.idEmpresa;

SELECT * FROM vista_solucionesBDconocimiento_empresa;
```

### *7.6. vista_total_tickets_empresas:*

```
-- VISTA DE TODOS LOS TICKETS ASIGNADOS POR EMPRESA ----------------------------------------------------
-- Esta vista muestra todos los tickets asignados a las empresas. Proporciona información sobre el número de ticket, el título del ticket,
-- el estado y el nombre de la empresa a la que están asignados. Esta vista utiliza un nivel de seguridad definido por el usuario admin@%.

CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `admin`@`%`
    SQL SECURITY DEFINER
VIEW `vista_total_tickets_empresas` AS
    SELECT
        `Tickets`.`idTicket` AS N°Ticket,
        Tickets.Titulo AS TituloTicket,
        Tickets.Estado AS Estado,
        Empresa.Nombre AS NombreEmpresa
    FROM
        ((Tickets
        JOIN Usuarios ON ((Tickets.idUsuarios = Usuarios.idUsuarios)))
        JOIN Empresa ON ((Usuarios.idEmpresa = Empresa.idEmpresa)))
```

# 8. Funciones

### 8.1. *F () PorcentajeTicketsporEmpresa:*

Función que calcula el porcentaje de los tickets creados por una empresa respecto al total (totas las empresas)
Ejemplo:

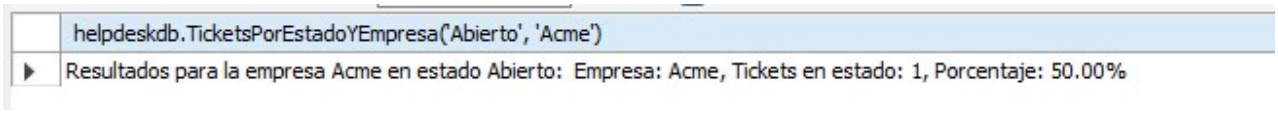
| | helpdeskdb.PorcentajeTicketsPorEmpresa('Deloitte') |
|---|---|
| ▶ | 11.11 |

```
-- Funcion que calcula el porcentaje de tickets relacionados con una empresa específica
-- en comparación con el total de tickets en la base de datos.

CREATE DEFINER=`admin`@`%` FUNCTION `PorcentajeTicketsPorEmpresa`(empresa_nombre VARCHAR(45)) RETURNS decimal(5,2)
    READS SQL DATA
    DETERMINISTIC
BEGIN
    DECLARE total_tickets INT;
    DECLARE tickets_empresa INT;
    DECLARE porcentaje DECIMAL(5, 2);
    -- Calcula el total de tickets
    SELECT COUNT(*) INTO total_tickets FROM Tickets;
    -- Calcula el total de tickets creados por la empresa especificada
    SELECT COUNT(*) INTO tickets_empresa FROM Tickets t
    INNER JOIN Usuarios u ON t.idUsuarios = u.idUsuarios
    INNER JOIN Empresa e ON u.idEmpresa = e.idEmpresa
    WHERE e.Nombre = empresa_nombre;

    -- Calcula el porcentaje
    IF total_tickets > 0 THEN
        SET porcentaje = (tickets_empresa / total_tickets) * 100;
    ELSE
        SET porcentaje = 0;
    END IF;
    RETURN porcentaje;
END
```

CODERHOUSE

8.2. ***F () TicketPorEstadoYEmpresa:***

Función que calcula el porcentaje de ticket en un estado especifico respecto a una empresa. Ejemplo:



| | helpdeskdb.TicketsPorEstadoYEmpresa('Abierto', 'Acme') |
|---|---|
| ▶ | Resultados para la empresa Acme en estado Abierto: Empresa: Acme, Tickets en estado: 1, Porcentaje: 50.00% |

```sql
-- Funcion que se utiliza para obtener información sobre los tickets de una empresa en un estado de ticket específico y calcular el porcentaje de tickets en ese estado.
CREATE DEFINER=`admin`@`%` FUNCTION `TicketsPorEstadoYEmpresa`(estado VARCHAR(45), empresa_nombre VARCHAR(45)) RETURNS varchar(200) CHARSET utf8mb4
    READS SQL DATA
    DETERMINISTIC
BEGIN
    DECLARE resultado VARCHAR(200);
    DECLARE total_tickets INT;
    DECLARE tickets_estado INT;
    DECLARE porcentaje DECIMAL(10, 2);
    -- Obtiene el total de tickets en estado especificado para la empresa especificada
    SELECT COUNT(*) INTO tickets_estado
    FROM Tickets
    INNER JOIN Usuarios ON Tickets.idUsuarios = Usuarios.idUsuarios
    INNER JOIN Empresa ON Usuarios.idEmpresa = Empresa.idEmpresa
    WHERE Tickets.Estado = estado AND Empresa.Nombre = empresa_nombre;
    -- Obtiene el total de tickets para la empresa especificada
    SELECT COUNT(*) INTO total_tickets
    FROM Tickets
    INNER JOIN Usuarios ON Tickets.idUsuarios = Usuarios.idUsuarios
    INNER JOIN Empresa ON Usuarios.idEmpresa = Empresa.idEmpresa
    WHERE Empresa.Nombre = empresa_nombre;

    -- Calcula el porcentaje
    IF total_tickets > 0 THEN
        SET porcentaje = (tickets_estado / total_tickets) * 100;
    ELSE
        SET porcentaje = 0;
    END IF;
    -- Construye el resultado
    SET resultado = CONCAT('Resultados para la empresa ', empresa_nombre, ' en estado ', estado, ': ', CHAR(10));
    SET resultado = CONCAT(resultado, 'Empresa: ', empresa_nombre, ', Tickets en estado: ', tickets_estado, ', Porcentaje: ', porcentaje, '%', CHAR(10));
    RETURN resultado;
END
```

CODERHOUSE

# 9. Procedimientos Almacenados

## 9.1. *sp_OrdenarUsuarios:*

Procedimiento almacenado que ordena un campo de la tabla “Usuarios” de manera ascendente o descendente según se especifique. Ejemplo:

```
call helpdeskdb.sp_OrdenarUsuarios('Area', 'ASC');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| idUsuarios | idEmpresa | Nombre | Apellido | Email | Rol | Genero | Area | Cargo | Fecha de creacion |
|---|---|---|---|---|---|---|---|---|---|
| 26 | 5 | NombreNuevo | ApellidoNuevo | carolina@example.com | RolNuevo | GeneroNuevo | AreaNueva | CargoNuevo | 2023-10-11 20:23:09 |
| 25 | 5 | Valeria | Ferrer | valeria.ferrer@example.com | Usuario | Femenino | Armas | Analista de Armas | 2023-10-11 00:56:30 |
| 24 | 5 | Pablo | Rodriguez | pablo.rodriguez@example.com | Usuario | Masculino | Armas | Analista de Armas | 2023-10-11 00:56:30 |
| 23 | 5 | Carolina | Díaz | carolina.diaz@example.com | Usuario | Femenino | Armas | Analista de Armas | 2023-10-11 00:56:30 |
| 21 | 5 | Fernanda | Gutierrez | fernanda.gutierrez@example.com | Administrador | Femenino | Armas | Gerente de Armas | 2023-10-11 00:56:30 |
| 22 | 5 | Martín | Lopez | martin.lopez@example.com | Tecnico | Masculino | Armas | Tecnico de Armas | 2023-10-11 00:56:30 |
| 17 | 4 | Elena | Perez | elena.perez@example.com | Tecnico | Femenino | Operaciones | Supervisor de Logística | 2023-10-11 00:56:30 |
| 16 | 4 | Fernando | Ramos | fernando.ramos@example.com | Administrador | Masculino | Operaciones | Director de Operaciones | 2023-10-11 00:56:30 |
| 18 | 4 | Jorge | Garrido | jorge.garrido@example.com | Usuario | Masculino | Operaciones | Analista de Logística | 2023-10-11 00:56:30 |
| 19 | 4 | Rosa | Jimenez | rosa.jimenez@example.com | Usuario | Femenino | Operaciones | Analista de Logística | 2023-10-11 00:56:30 |
| 20 | 4 | Alejandro | Vargas | alejandro.vargas@example.com | Usuario | Masculino | Operaciones | Analista de Logística | 2023-10-11 00:56:30 |

```
-- Procedimiento almacenado que ordena un campo de la tabla " Usuarios" de forma acendente o decendente.
--  Toma 2 parametros de entrada : " p_campoOrdenamiento" y " p_orden" y ordena la tabla de forma acendente o decendente

CREATE DEFINER=`admin`@`%` PROCEDURE `sp_OrdenarUsuarios`(
    IN p_campoOrdenamiento VARCHAR(45),
    IN p_orden VARCHAR(10)
)
BEGIN
    SET @query = CONCAT('SELECT * FROM Usuarios ORDER BY `', p_campoOrdenamiento, '` ', p_orden, ';');
    PREPARE stmt FROM @query;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END

-- EJEMPLOS
-- CALL sp_OrdenarUsuarios('Nombre', 'ASC');
-- CALL sp_OrdenarUsuarios('Apellido', 'DESC');
```

### 9.2. *Sp_RegistrarSolicitud:*

Procedimiento almacenado para registrar nuevo ticket, tomando como parámetros de entrada los datos mínimos para la generación del ticket. Devuelve el numero de ticket creado. Ejemplo:

```
call helpdeskdb.sp_RegistrarSolicitud(13, 'Mouse defectuoso , se solicita cambio', 2, 'Incidente', 'Alta', 'Cambio de mouse');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| idTicket |
|---|
| 21 |

```
-- Procedimiento almacenado que registra una nuevo ticket en la base de datos.
-- Toma como parametros el ID del usuario que crea el ticket, el titulo, la descripcion , la categoria , el tipo y la prioridad
-- Devuelve un nuevo ticket con los datos ingresados como parametros de entrada

CREATE DEFINER=`admin`@`%` PROCEDURE `sp_RegistrarSolicitud`(
    IN p_idUsuario INT,
    IN p_descripcionSolicitud TEXT,
    IN p_idCategoria INT,
    IN p_tipo VARCHAR(45),
    IN p_prioridad VARCHAR(45),
    IN p_titulo VARCHAR(255)
)
BEGIN
    -- Insertar la solicitud en la tabla de tickets
    INSERT INTO Tickets (idUsuarios, idCategoria, `Fecha de creacion`, Tipo, Prioridad, descripcion, Estado, Titulo)
    VALUES (p_idUsuario, p_idCategoria, NOW(), p_tipo, p_prioridad, p_descripcionSolicitud, 'Abierto', p_titulo);

    -- Obtener el ID del ticket recién creado
    SET @idTicket = LAST_INSERT_ID();

    -- Registrar la solicitud en el registro de comentarios
    INSERT INTO Comentarios (idUsuarios,`Fecha y hora`,texto_Comentario)
    VALUES (p_idUsuario, NOW(), CONCAT('Solicitud creada: ', p_descripcionSolicitud));

    -- Devolver el ID del ticket recién creado
    SELECT @idTicket AS idTicket;
END
```

# 10. Triggers

Se implementaron triggers para automatizar acciones en la base de datos cuando ocurren eventos específicos en las tablas. Respecto a la creación de los trigger se crearon 2 trigger after y 2 before para las tablas Tickets y Usuarios:

## *10.1. TriggerTickets_AfterUpdate*

Se desencadena después de la actualización de algún campo de la tabla “ticket”. Registra actualizaciones de estos estados en la tabla “Ticket_Log”.

```
CREATE TABLE Ticket_Log (
    idLog INT AUTO_INCREMENT,
    idTicket INT,
    FechaHoraCambio DATETIME,
    Cambio VARCHAR(255),
    PRIMARY KEY (idLog)
);
DELIMITER //
CREATE TRIGGER Tickets_AfterUpdate
AFTER UPDATE ON Tickets
FOR EACH ROW
BEGIN
    IF OLD.Estado <> NEW.Estado THEN
        INSERT INTO Ticket_Log (idTicket, Cambio, FechaHoraCambio)
        VALUES (NEW.idTicket, CONCAT('Estado actualizado de "', OLD.Estado, '" a "', NEW.Estado, '"'), NOW());
    END IF;
END;
```

## 10.2. *TriggerTickets_BeforeInsert:*

Se crea condición antes de la generación de un ticket. La condición consiste en que los únicos valores validos para el campo “Estado” de la tabla tickets son: Abierto, Trabajo en Curso, Pendiente Información y Cerrado.

```
CREATE TRIGGER Tickets_BeforeInsert
BEFORE INSERT ON Tickets
FOR EACH ROW
BEGIN
    DECLARE estado_valido INT;
    SET estado_valido = 0;

    IF NEW.Estado = 'Abierto' OR NEW.Estado = 'Trabajo en Curso' OR NEW.Estado = 'Pendiente Informacion' OR NEW.Estado = 'Cerrado' THEN
        SET estado_valido = 1;
    END IF;

    IF estado_valido = 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'El estado no es válido. Debe ser Abierto, Trabajo en Curso, Pendiente Información o Cerrado.';
    END IF;
END;
//
DELIMITER ;

-- Prueba de trigger con creacion de tickets

INSERT INTO Tickets (idUsuarios, idCategoria, `Fecha de creacion`, Tipo, Prioridad, descripcion, resolucion, Estado, `Fecha cambio estado`, Titulo)
VALUES (8, 3, NOW(), 'Incidente', 'Alta', 'TEST TRIGGER', 'TEST TRIGGER', 'Abierto123', NOW(), 'TEST TRIGGER');

 -- Prueba de trigger con actualizacion de estado de ticket

 UPDATE Tickets
SET Estado = 'Casi resuelto'
WHERE idTicket = 15;
```

## 10.3. *TriggerUsuarios_AfterUpdate*

Se desencadena después de la actualización de algún campo de la tabla “usuarios”. Registra actualizaciones de estos estados en la tabla “Usuarios_Log”.

```
-- Creacion de Tabla Usuarios_Log --
CREATE TABLE Usuarios_Log (
    idLog INT AUTO_INCREMENT,
    idUsuarios INT,
    FechaHoraCambio DATETIME,
    Cambio VARCHAR(255),
    PRIMARY KEY (idLog)
);
-- Registra actualizaciones en tabla "Usuarios_Log ". Cualquier actualizacion de Usuarios se registra en nueva tabla  ---
DELIMITER //
CREATE TRIGGER Usuarios_AfterUpdate
AFTER UPDATE ON Usuarios
FOR EACH ROW
BEGIN
    DECLARE cambio VARCHAR(255);

    SET cambio = '';

    IF OLD.Nombre <> NEW.Nombre THEN
        SET cambio = CONCAT(cambio, 'Nombre actualizado de "', OLD.Nombre, '" a "', NEW.Nombre, '"; ');
    END IF;

    IF OLD.Apellido <> NEW.Apellido THEN
        SET cambio = CONCAT(cambio, 'Apellido actualizado de "', OLD.Apellido, '" a "', NEW.Apellido, '"; ');
    END IF;

    IF OLD.Email <> NEW.Email THEN
        SET cambio = CONCAT(cambio, 'Email actualizado de "', OLD.Email, '" a "', NEW.Email, '"; ');
    END IF;
    IF OLD.Rol <> NEW.Rol THEN
        SET cambio = CONCAT(cambio, 'Rol actualizado de "', OLD.Rol, '" a "', NEW.Rol, '"; ');
    END IF;

    IF OLD.Genero <> NEW.Genero THEN
        SET cambio = CONCAT(cambio, 'Género actualizado de "', OLD.Genero, '" a "', NEW.Genero, '"; ');
    END IF;

    IF OLD.Area <> NEW.Area THEN
        SET cambio = CONCAT(cambio, 'Área actualizada de "', OLD.Area, '" a "', NEW.Area, '"; ');
    END IF;

    IF OLD.Cargo <> NEW.Cargo THEN
        SET cambio = CONCAT(cambio, 'Cargo actualizado de "', OLD.Cargo, '" a "', NEW.Cargo, '"; ');
    END IF;

    -- Agrega más condiciones IF para otros campos que desees registrar

    IF cambio <> '' THEN
        INSERT INTO Usuarios_Log (idUsuarios, Cambio, FechaHoraCambio)
        VALUES (NEW.idUsuarios, cambio, NOW());
    END IF;
END;
//
DELIMITER ;

-- Actualizacion de prueba
UPDATE Usuarios
SET Nombre = 'NuevoNombre', Apellido = 'NuevoApellido' , Email='NuevoEmail@email.com'
WHERE idUsuarios = 10;
```

### 10.4. *TriggerUsuarios_BeforeInsert*

Se crea condición antes de la generación de un usuario. La condición consiste en si el email del usuario nuevo ya existe, no puede agregar el registro. A su vez si se cumple esta condición arroja el mensaje: “No se puede crear el usuario porque el correo ya existe en otro usuario”.

```
DELIMITER //
CREATE TRIGGER Usuarios_BeforeInsert
BEFORE INSERT ON Usuarios
FOR EACH ROW
BEGIN
    -- Verificar si el correo electrónico ya existe en la tabla
    IF EXISTS (SELECT 1 FROM Usuarios WHERE Email = NEW.Email) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No se puede crear el usuario porque el correo ya existe en otro usuario.';
    END IF;
END;
```