

Sistema de gestión de incidentes

La siguiente presentación documenta las tareas realizadas para el armado y configuración de una BD que busca manejar incidentes para un equipo de trabajo.

Jejer Gustavo Javier

Objetivo

- ▶ Crear una BD consistente que pueda almacenar datos sobre incidentes que deben trabajar los equipos de TI, como así también la información histórica y una medición de los tiempos (KPI).

Situación problemática

- ▶ Actualmente no se cuenta con una herramienta interna que pueda canalizar los tickets que los usuarios envían a TI por mail, por ello surge la búsqueda del desarrollo interno de una APP para la canalización y gestión de incidentes.

Script de definición de BD

- ▶ Repositorio con todo el código utilizado:
<https://github.com/Krieger93/IncidentesDB>
- ▶ Script completo de BD:
https://github.com/Krieger93/IncidentesDB/blob/master/incidentesDB_full_script.sql

Definición de tablas y campos

USUARIOS	CONTIENE A LOS USUARIOS NORMALES Y ADMINS QUE GENERAN INCIDENTES EN BASE A UN PROBLEMA.
CATEGORIAS	LAS CATEGORIAS DESCRIBEN/ENGLOBAN EL TIPO DE PROBLEMA DEL INCIDENTE.
INCIDENTES	AQUÍ SE ALMACENA LA MAYOR PARTE DE LA INFORMACION DE LOS INCIDENTES CREADOS POR LOS USUARIOS/ADMINS.
GRUPOS	SE DEFINE A GRUPOS COMO UN EQUIPO DE TRABAJO LOS CUALES PUEDEN ATENDER DIFERENTES PROBLEMATICAS.
KPIS	ALMACENA LAS MEDICIONES DE TIEMPO DE RESOLUCION DE LOS INCIDENTES Y LA VALORACION DE LOS USUARIOS.
BKPINCIDENTES	POR CADA INSERT DE UN NUEVO INCIDENTE, ESTE SE GUARDA EN ESTA TABLA SECUNDARIA PARA MAYOR SEGURIDAD Y RESPALDO.
BKPUERS	LOS USUARIOS ELIMINADOS SON ALMACENADOS AQUÍ.

LLAVE	TABLA USUARIOS	TIPO DE DATO
PK	id_user	INT
	user	VARCHAR(50)
	pass_user	VARCHAR(25)
	mail_user	VARCHAR(50)
	nivel_user	INT
FK	id_grupo	INT

LLAVE	TABLA CATEGORIAS	TIPO DE DATO
PK	id_categoria	INT
	titulo_categoria	VARCHAR(50)

LLAVE	TABLA KPIS	TIPO DE DATO
PK	id_kpi	INT
FK	id_incidente	INT
	tiempo_resolucion	DATE
	primera_respuesta	DATE
	puntaje_usuario	DATE

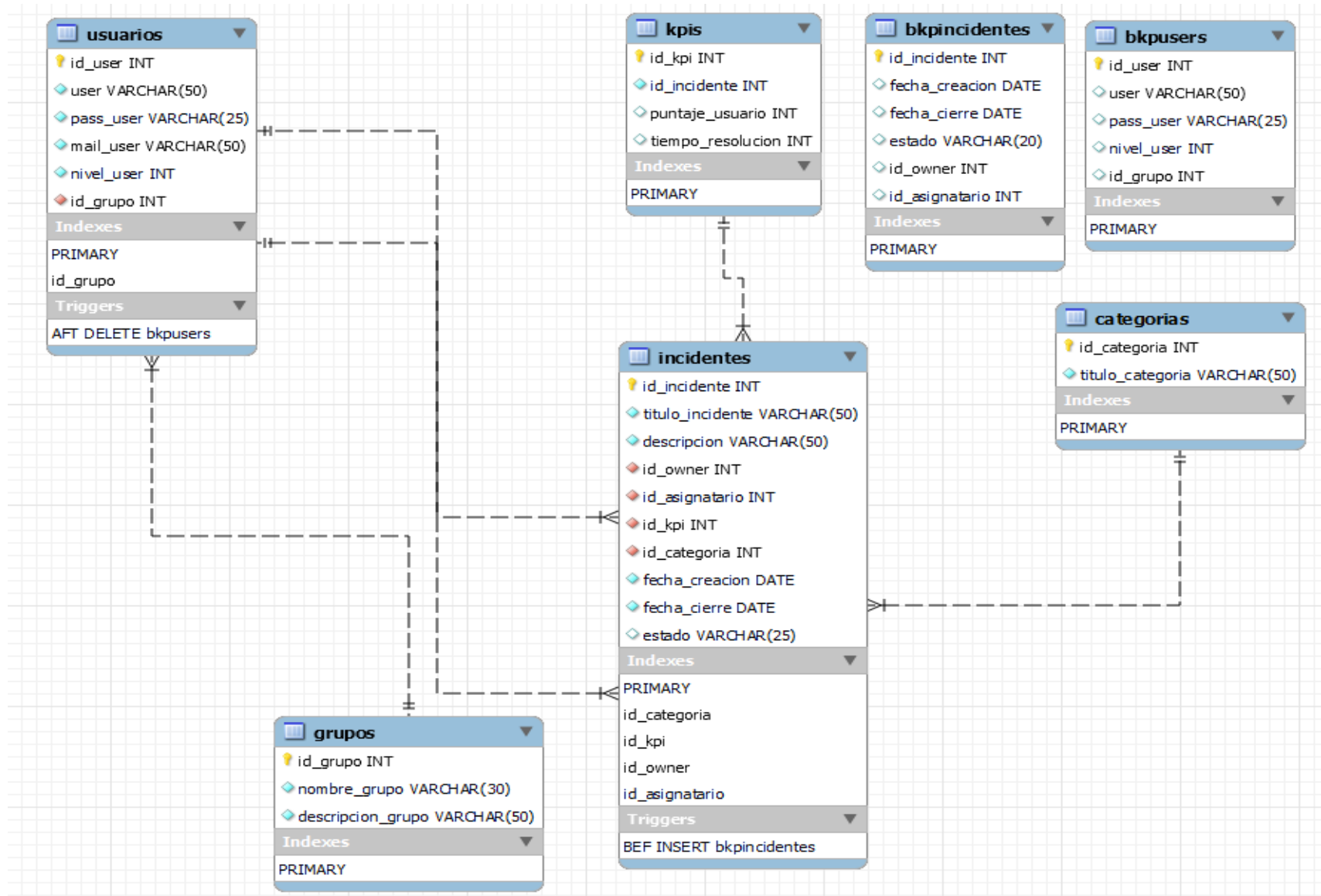
LLAVE	TABLA BKPUERS	TIPO DE DATO
PK	id_user	INT
	user	VARCHAR(50)
	pass_user	VARCHAR(25)
	nivel_user	INT
	id_grupo	INT

LLAVE	TABLA INCIDENTES	TIPO DE DATO
PK	id_incidente	INT
	titulo_incidente	VARCHAR(50)
	descripcion	VARCHAR(50)
FK	id_owner	INT
FK	id_asignatario	INT
FK	id_kpi	INT
	fecha_creacion	DATE
	fecha_cierre	DATE
FK	id_categoria	INT

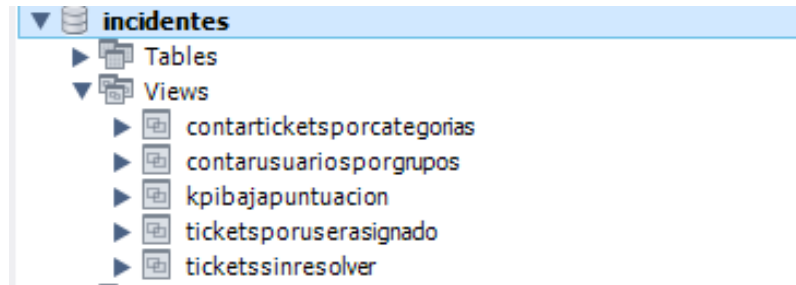
LLAVE	TABLA GRUPOS	TIPO DE DATO
PK	id_grupo	INT
	nombre_grupo	VARCHAR(30)
	descripcion_grupo	VARCHAR(50)

LLAVE	TABLA BKPINCIDENTES	TIPO DE DATO
PK	id_incidente	INT
	fecha_creacion	DATE
	fecha_cierre	DATE
	estado	VARCHAR(20)
	id_owner	INT
	id_asignatario	INT

Estructura de Base de Datos



Vistas I



- ▶ **Contar tickets por categoría:** Devuelve el total de tickets que tiene asignada cada categoría. [Tablas: incidentes, categorias]
- ▶ **Contar usuarios por grupos:** Devuelve la cantidad de usuarios que tienen los grupos existentes. [Tablas: usuarios y grupos]
- ▶ **KPI baja puntuación:** Devuelve aquellos incidentes donde el puntaje del usuario es menor a lo esperado. [Tabla: KPIS]
- ▶ **Ticket por usuario asignado:** Join que devuelve los usuarios y sus tickets asignados. [Tablas: incidentes y usuarios]
- ▶ **Tickets sin resolver:** Devuelve el listado de tickets que tengan cualquier estado excepto “Solved”. [Tabla: incidentes]

Vistas II

► Contar tickets por categoría:

```
1 • SELECT * FROM incidentes.contarticketsporcategorias;
```

Result Grid			
		Filter Rows:	
		Export:	
		Wrap Cell Content:	
	id_categoria	titulo_categoria	TicketsxCategoria
1	1	WINDOWS	2
2	2	LINUX	1
3	3	BASE DE DATOS	1

Vistas III

► Contar usuarios por grupos:

```
1 • | SELECT * FROM incidentes.contarusuariosporgrupos;
```

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
nombre_grupo	id_grupo	UsuariosxGrupo			
DevOps	1	1			
DBA	2	1			
Networking	3	1			
Customers	4	1			

Vistas IIII

► KPI baja puntuación:



```
1 • |SELECT * FROM incidentes.kpibajapuntuacion;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	puntaje_usuario			
	5			
	2			
	3			

Vistas V

► Ticket por usuario asignado:

```
1 • SELECT * FROM incidentes.ticketsporusuarioasignado;
```

Result Grid				
Filter Rows: <input type="text"/>				
Export: 				
Wrap Cell Content: 				
titulo_incidente	descripcion	estado	id_user	user
► probando trigger	probando si se copia a tabla de bkp	Nuevo	1	gjejer
Replicar DB a PROD	Pase a PROD de BD	Solved	1	gjejer
Windows VM falla	Servicios fallando	Solved	4	normaluser
Linux VM falla	La VM no responde	Validacion	4	normaluser

Vistas VI

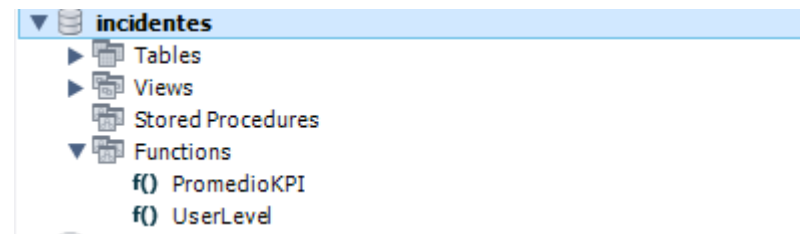
► Tickets sin resolver:

```
1 • SELECT * FROM incidentes.ticketssinresolver;
```

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
titulo_incidente	descripcion	estado			
Linux VM falla	La VM no responde	Validacion			
probando trigger	probando si se copia a tabla de bkp	Nuevo			

Funciones

- ▶ **Promedio KPI:** Devuelve el promedio del puntaje otorgado por cada usuario en los incidentes.
- ▶ **User level:** Ingresando el tipo de nivel del usuario nos devuelve si este es admin o no.



Funciones I

► Promedio KPI:

```
1 • select incidentes.PromedioKPI();  
2
```

Result Grid		Filter Rows:	Export:
	incidentes.PromedioKPI()		
►	3.33333		

Funciones II

► User level:

```
1 • select incidentes.UserLevel(0);  
2 |
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	incidentes.UserLevel(0)			
ES ADMIN				

Stored Procedures

- ▶ **SP_ordenarinc:** Ordena la tabla incidentes en base a la columna dada y al orden solicitado.
- ▶ **SP_updateinestado:** Actualiza el estado de un incidente
- ▶ [REPO](#)

```
SP_ordenarinc

1 • CREATE DEFINER='root'@'localhost' PROCEDURE `SP_ordenarinc`(IN campo VARCHAR(30), IN orden VARCHAR(4))
2 BEGIN
3     -- DECLARE clausula varchar(50);
4     DECLARE clausula varchar(50);
5     -- @clausula ES LA CLAUSULA FINAL A EJECUTAR, EN LA QUE SE ENCADENAN LA SENTENCIA SQL Y LAS VARIABLES
6     SET @clausula = concat('SELECT * FROM incidentes ORDER BY ', campo, ' ', orden);
7     -- SE PREPARA, EJECUTA Y CIERRA LA SENTENCIA
8     PREPARE ejecutar FROM @clausula;
9     EXECUTE ejecutar;
10    DEALLOCATE PREPARE ejecutar;
11 END
```

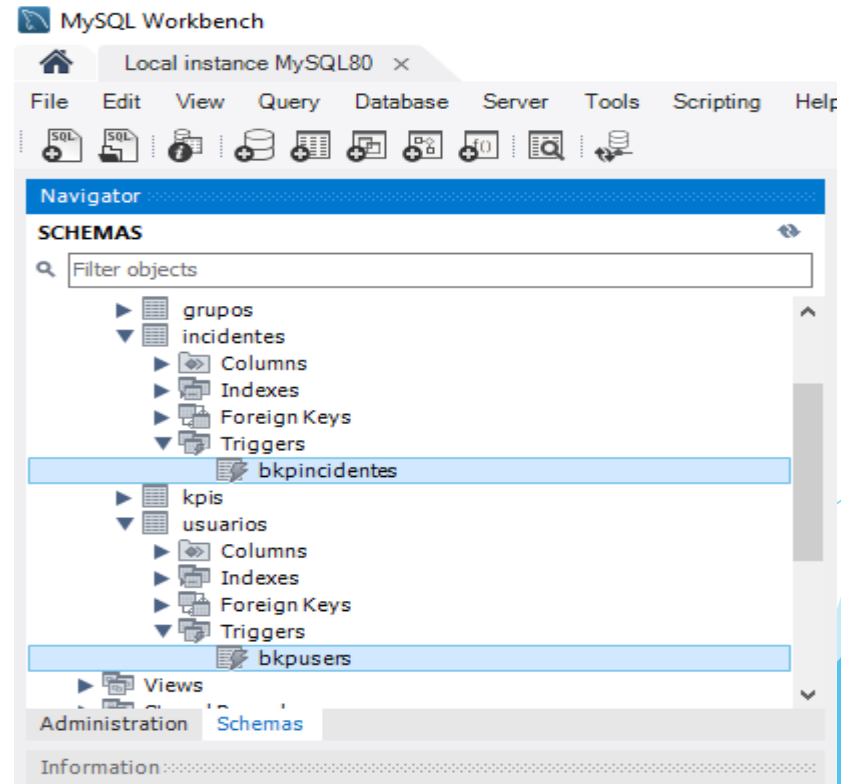
```
SP_updateinestado

1 • CREATE DEFINER='root'@'localhost' PROCEDURE `SP_updateinestado`(IN i_id INT, IN i_estado varchar(25))
2 BEGIN
3     UPDATE incidentes
4     SET
5     estado = i_estado WHERE id_incidente = i_id;
6 END
```


Triggers

- ▶ Se generaron dos Triggers los cuales se ejecutan luego de una inserción y otro que se ejecuta luego de una eliminación de datos, para ello se generaron dos tablas (**bkpincidentes** y **bkpusers**) que servirán de contingencia brindándole mayor fiabilidad a la BD.

[Repo Triggers](#)



Triggers II: bkpincidentes

► Test de trigger de inserción de datos

```
1  -- TEST del trigger
2  • INSERT INTO incidentes(id_incidente, titulo_incidente, descripcion, id_owner, id_asignatario, id_kpi, id_categoria, fecha_creacion, fecha_cierre, estado)
3  values(15, 'test_trigger', 'trigger_test', 1, 2, 3, 1, '2022-02-12', '2022-03-06', 'Nuevo');
4
5  • SELECT *
6  FROM bkpincidentes;
7
8  -- Agregamos un nuevo usuario a borrar
9  • INSERT INTO usuarios(user, pass_user, mail_user, nivel_user, id_grupo)
10 values('testinguser', 'qpwoejq', 'opqwe@gmail.com', 3, 4);
11
12 -- Quitamos la configuracion de seguridad que por defecto viene e impide eliminar valores de las tablas
13 • SET SQL_SAFE_UPDATES = 0;
14
15
16 -- TEST del trigger
17 • delete from usuarios
18 where nivel_user = 3;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	id_incidente	fecha_creacion	fecha_cierre	estado	id_owner	id_asignatario
▶	0	2022-01-12	2022-01-20	Nuevo	1	2
	15	2022-02-12	2022-03-06	Nuevo	1	2
*	NULL	NULL	NULL	NULL	NULL	NULL

Triggers III: bkpusers

► Test trigger previo a eliminación de usuario

```
8  -- Agregamos un nuevo usuario a borrar
9  • INSERT INTO usuarios(user, pass_user, mail_user, nivel_user, id_grupo)
10 values('usernotfound', 'error404', 'asereje@jadeje.com', 3, 4);
11
12 • SELECT *
13 FROM usuarios;
14
15 -- Quitamos la configuracion de seguridad que por defecto viene e impide eliminar valores de las tablas
16 • SET SQL_SAFE_UPDATES = 0;
17
18
19 -- TEST del trigger
20 • delete from usuarios
21 where nivel_user = 3;
22
23 • SELECT *
24 FROM bkpusers;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	id_user	user	pass_user	nivel_user	id_grupo
5		testinguser	qpwoejq	3	4
6		testinguser	qpwoejq	3	4
7		usernotfound	error404	3	4
8	NULL	NULL	NULL	NULL	NULL

Manejo de usuarios

- A modo de ejemplo se generaron 2 nuevos usuarios:

```
1 • use mysql;
2
3 -- Creo el primer usuario
4 • CREATE USER 'prueba404'@'localhost' identified BY 'test1';
5
6 -- Creo el segundo usuario
7 • CREATE USER 'prueba405'@'localhost' identified BY 'test2';
8
9
10 -- Agrego permisos solo de lectura al primer usuario
11 • GRANT SELECT ON incidentes.* TO 'prueba404'@'localhost';
12
13 -- Agrego permisos de lectura, inserción y modificación al segundo usuario
14 • GRANT SELECT, INSERT, UPDATE ON incidentes.* TO 'prueba405'@'localhost';
```

```
mysql> SELECT user, host, account_locked, password_expired FROM user;
```

user	host	account_locked	password_expired
mysql.infoschema	localhost	Y	N
mysql.session	localhost	Y	N
mysql.sys	localhost	Y	N
prueba	localhost	N	N
prueba1	localhost	N	N
prueba404	localhost	N	N
prueba405	localhost	N	N
root	localhost	N	N

8 rows in set (0.00 sec)

```
mysql> use mysql
Database changed
mysql> select user from user;
```

user
mysql.infoschema
mysql.session
mysql.sys
prueba
prueba1
prueba404
prueba405
root

8 rows in set (0.00 sec)

Backup BD Incidentes

- ▶ La siguiente tarea de backup se llevo adelante mediante la toma de los datos de las tablas bkpincidentes, bkpusers, incidentes y usuarios dejando de lado el Schema de la BD.
- ▶ LOG BACKUP

Herramientas y tecnologías utilizadas

- ▶ Interfaz web para la interacción con la BD (HTML, CSS, JS)
- ▶ GIT y GitHub para el almacenamiento y orden de los scripts de la BD
- ▶ Motor de BD mysql y workbench.